NÚMERO 9



DIV manía

995 ptas.
PORTUGAL 990 ESC (CONT) 5,98 €
www.prensatecnica.com
Año 2 • Número 9

• M.U.G.E.N.
Crea luchadores para tus juegos

• Scripts
Aprende a programar estas utilidades

• Poser
Aplicando texturas a tus juegos

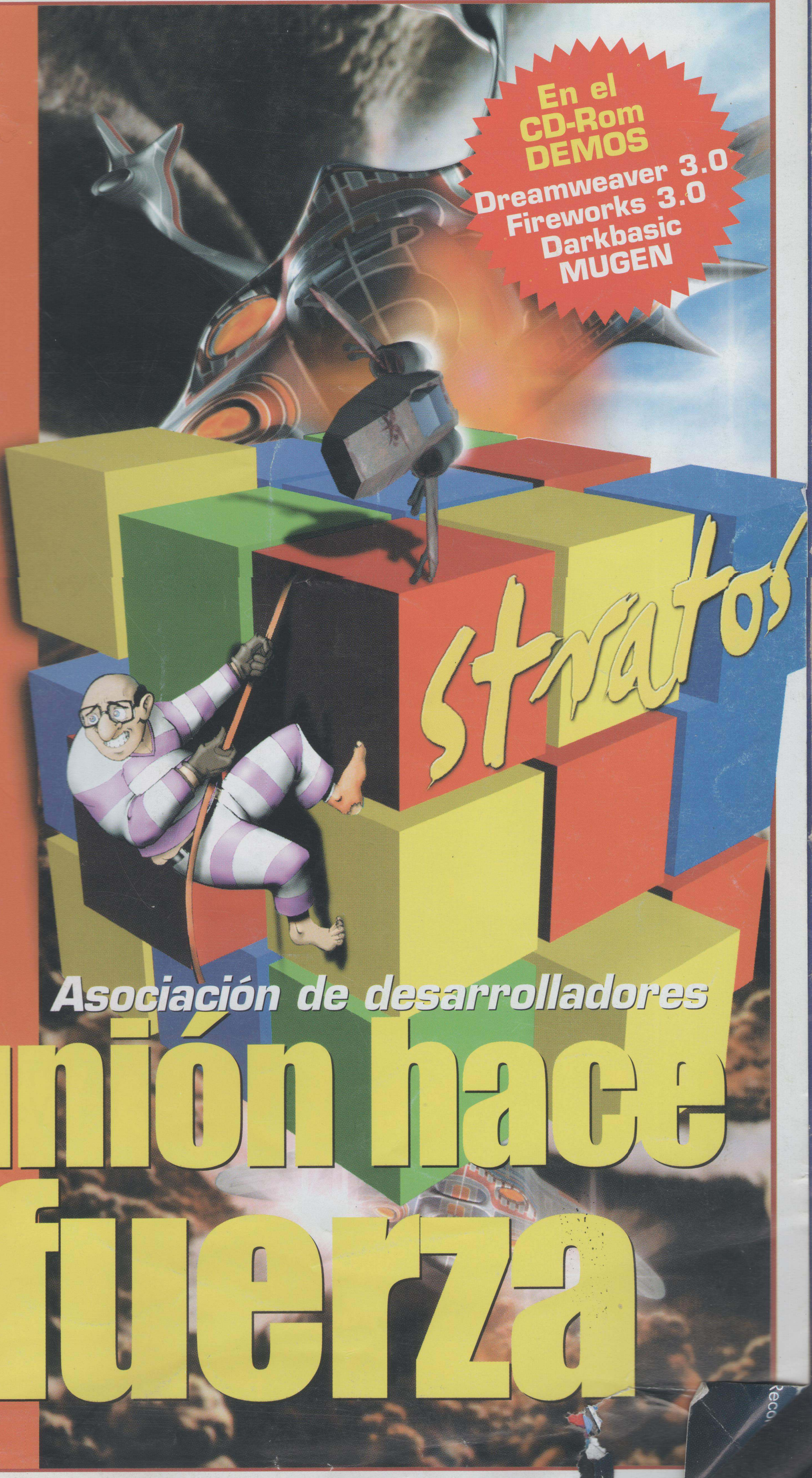
• RPG
Vistiendo y equipando personajes

• Acción plataformas
Nos metemos con los mapeados tiles

• 3D
Seguimos haciendo nuestro juego

• Iniciación DIV
Trabajando con cadenas

• DIV interno
Creando un generador de código



En el CD-Rom DEMOS
Dreamweaver 3.0
Fireworks 3.0
Darkbasic
MUGEN

Asociación de desarrolladores

La unión hace la fuerza

DIV MANIA
Prensa Técnica
8 413042 087533 00009



que con esta función podremos crearlos en tiempo de ejecución.

Otro truco relacionado con el trato de un elemento de manera distinta a la que veníamos haciendo hasta ahora, es el que veremos a continuación, que tiene que ver con los gráficos y la pantalla. Ahora, por ejemplo, se pueden hacer capturas mediante programación, además, también podemos grabar mapas en el disco duro e incluso crear nuevos mapas. Podremos, por ejemplo, hacer una captura de pantalla, y luego trabajar con ella, como si fuera otro gráfico más del fichero tipo fpg. Esto nos da unas grandes posibilidades a la hora de hacer fundidos de pantalla. Un caso práctico sería, el capturar la pantalla, para luego borrarla, poner nuestro gráfico creado, y redimensionarlo, con lo que conseguiríamos un efecto de fundido muy curioso.

Cada paterna consta normalmente de 64 líneas

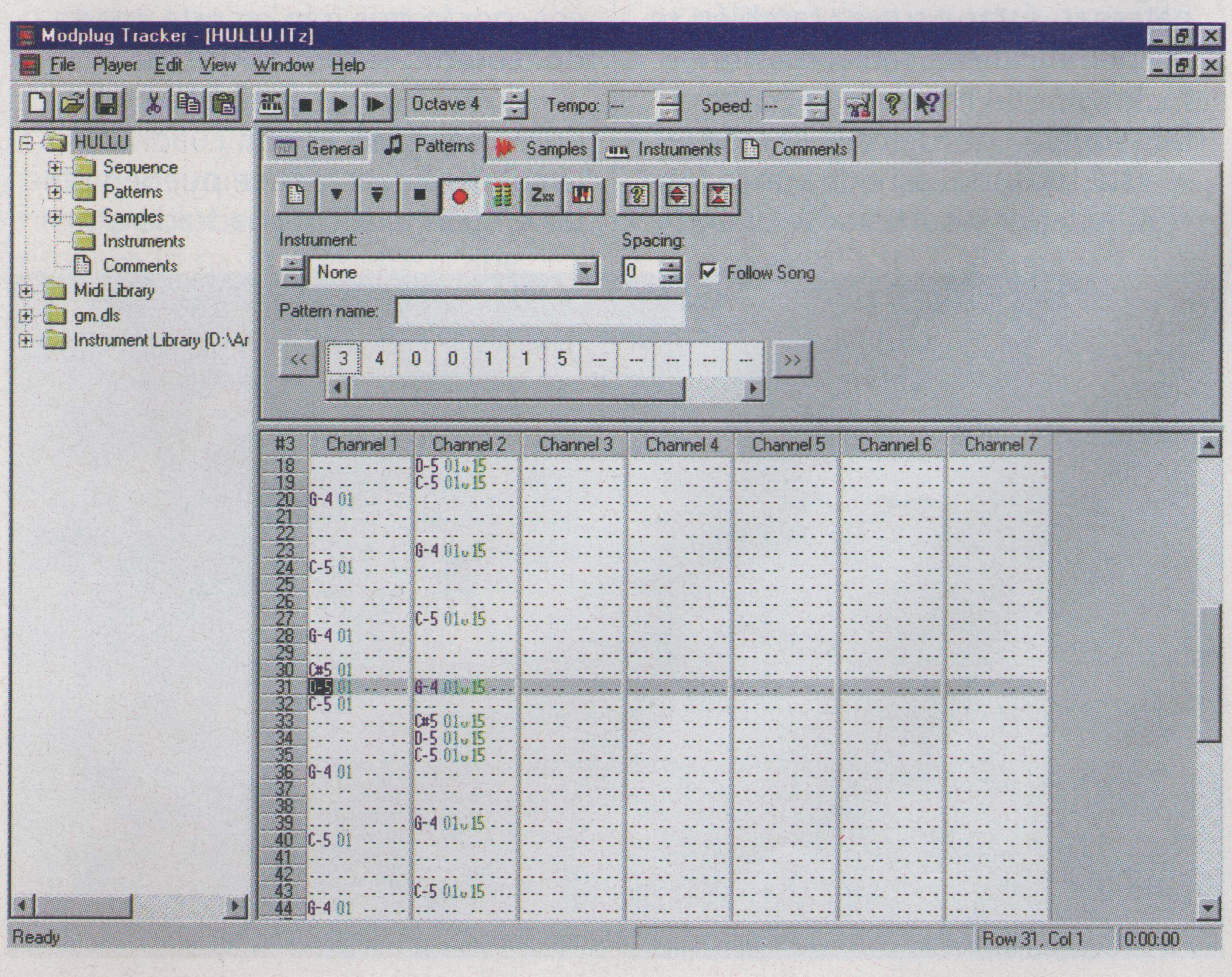
También podremos crear un gráfico desde cero, usando otras funciones de DIV para pintar puntos, o bloques gráficos. Esto, unido a la posibilidad de poder grabar cualquier gráfico en el disco, nos permite hacer capturas de pantallas o crear y grabar gráficos creados por el usuario. Además, se le puede asignar la pantalla, o un gráfico creado por nosotros a un proceso determinado, para luego tratarlo como tal, al igual que hacíamos con los textos, y reescalarlo, moverlo, girarlo, etc.

Con esto quedan vistos casi todos los trucos relacionados con gráficos. Hay que resaltar que la posibilidad de grabar gráficos en el disco, y crear nuevos gráficos en memoria para su posterior utilización es una gran mejora, ya que antes, al no disponer de estos medios el programador estaba limitado al fichero fpg o mapa que cargara desde disco, y no podía grabar nada en el disco, por lo que no tenía ningún dispositivo de salida. Ahora veremos las funciones relacionadas con primitivas gráficos, y creación y manipulación de gráficos creados mediante programación.

– *int draw(tipo,color,porcentaje, region,x0,y0,x1,y1):* esta función es la primera de las relacionadas con las primitivas. Con ella podremos dibujar cualquier tipo de primitiva, siendo éste el valor del primer parámetro. Los valores serán uno para línea recta, dos y tres para Rectángulo y Rectángulo relleno y, por último, tres y cuatro para utilizar una Elipse o Elipse rellena. El segundo parámetro que recibe la función es el color de dicha primitiva dentro de la paleta. Luego se debe indicar un porcentaje de transparencia de la primitiva, este valor podrá variar entre 0 y 15. También se indicará la región de pantalla donde aparecerá la primitiva, y la zona cuadrangular donde será pintada. Es decir, cualquier primitiva, tanto líneas, como elipses o rectángulos, se pueden definir mediante dos puntos, que son los últimos cuatro parámetros. La función devuelve un código, que identificará a la primitiva, para luego poder borrarla o mover
– *int move_draw(identificador, color, porcentaje, x0, y0, x1, y1):* con esta función podremos mover o cambiar el color, el tamaño o el porcentaje de transparencia. Para ello, debemos indicar como primer parámetro el identificador de la primitiva, además de los nuevos parámetros que deseemos.
– *int delete_draw(identificador):* y ésta es la última función dedicada a las primitivas, con ella borraremos tanto de la pantalla, como de la memoria, la primitiva que hemos creado. Para ello, deberemos indicar el identificador apropiado, devuelto por la rutina de creación.
– *int new_map(ancho,alto,centro x, centro y, color):* con esta función podremos crear un gráfico nuevo en memoria. Funciona de forma parecida a como lo hace la función *load_map()*, devolviendo un código identificador de mapa, similar al de esta función. Como parámetros, debemos indicar el ancho y alto del gráfico, así como las coordenadas del centro del gráfico y el color de fondo con el que se rellenará el mapa.
– *int screen_copy(región, fichero, grafico,x,y,ancho,alto):* con esta función conseguiremos copiar un trozo del contenido actual de una región de pantalla a un gráfico de un fichero determinado. Los parámetros son la región elegida, el código de fichero y gráfico donde se guardará el gráfico, y las coordenadas iniciales y tamaño del trozo de pantalla a copiar.
– *int save_map(fichero, grafico, nombre archivo):* si queremos hacer una captura de pantalla, con la función anterior y con ésta, podremos hacerlo, ya que permite grabar un gráfico en el disco duro. Tendremos que indicar el fichero y gráfico elegidos, así como el nombre del archivo dentro del disco duro.
– *int write_in_map(fuente, texto, centrado):* por último, con esta función podremos convertir un texto en un mapa gráfico. Los parámetros utilizados son el código de fuente, el texto elegido y el código de centrado, que es similar al usado con las funciones *write()* y *write_int()*. Devolverá un código de gráfico, similar al de las funciones *load_map()* y *new_map()*.

Funciones de memoria y sistema

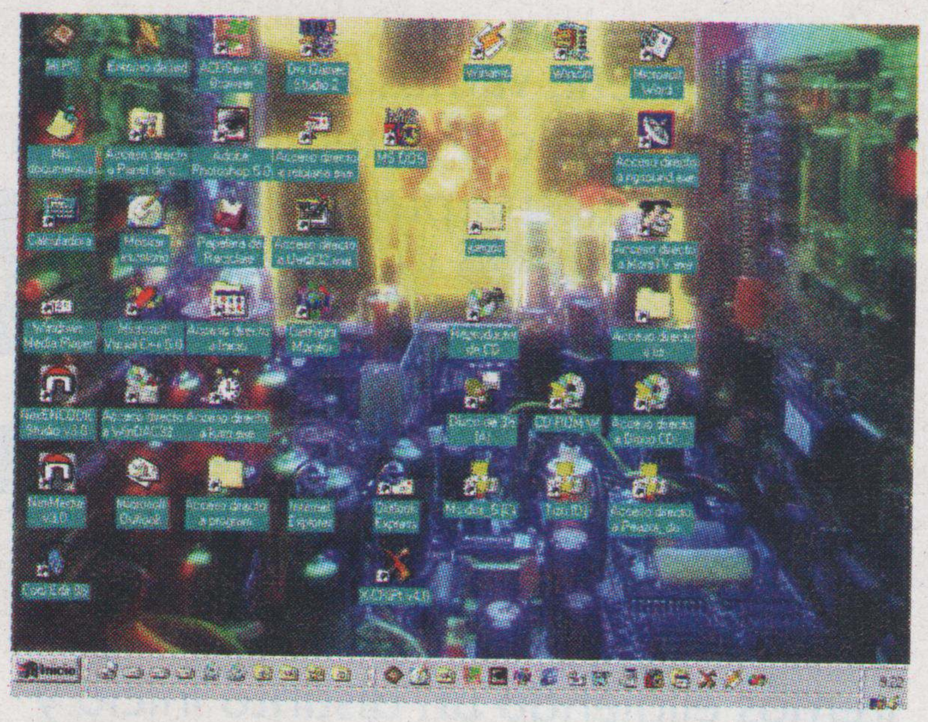
En este apartado veremos un par de funciones que, combinadas con otras, nos permiten describir un par de trucos. La primera función tiene que ver con la memoria, se trata de



la función *memory_free()*, que nos permite saber cuánta memoria hay libre, o lo que es lo mismo, memoria sin usar. Pero esta función por sí misma, no tiene ninguna utilidad, ya que el programador debe tener información sobre este dato. Existen numerosas formas de conocerlo, la primera es usando el debug de DIV, pulsando la tecla F12 durante el juego. También podemos imprimir en la pantalla este dato, para así visualizarlo, aunque el mejor método es hacer lo que denominamos comúnmente con un fichero log. Es decir, durante el transcurso del juego vamos comprobando la memoria libre, y el valor devuelto lo vamos grabando en un fichero en el disco, que luego podremos revisar tranquilamente, para comprobar si nuestro programa pierde memoria en alguna acción realizada.

Ahora pasemos al otro truco, del que ya vimos una versión, que se podría denominar ampliada. Se trata de usar la función *ignore_error()*. Dentro de las opciones de compilación, existe una que permite ignorar todos los errores que aparezcan en el juego. Esto puede ser hasta cierto punto peligroso, ya que puede producir ejecuciones del programa y efectos muy extraños, y no deseados. Si nuestro caso es el de que únicamente tenemos un error determinado dentro del programa, podremos usar la función *ignore_error()*, que únicamente ignora un error determinado, dejando que aparezcan los demás errores. Esto puede ser útil en determinados casos, aunque tanto la utilización de la opción de compilación, como de esta función, denota un estilo de programación un tanto pobre, no siendo recomendable su uso. Aunque siempre hay casos, en que es totalmente necesario, sobre todo, con los modos 8.

Otro aspecto, que aunque no tiene ninguna función relacionada, sí tiene que ver algo con el sistema es la creación de una nueva palabra reservada. Me explico, antes todos los procesos, eran eso, procesos, comenzaban con la palabra reservada *PROCESS*, debían tener normalmente un *FRAME*, para aparecer en pantalla, además de devolver el código identificador del proceso. También podíamos, retornar un valor propio pero, en este caso, no podíamos usar la palabra reservada *FRAME*, ya que dicho proceso no podía aparecer en pantalla. En ese caso, dicho proceso funcionaba, más que como un proceso, como una función. Ahora con DIV2, podremos indicar que el proceso es una función, para así clarificar de mejor manera los listados, usando la palabra reservada *FUNCTION*. Así, podremos separar, y tener claro qué parte de código es un proceso, y cuál es una función. Lógicamente, este tipo de funciones no podrá tener la palabra *FRAME*, además de retornar un valor. También podrá recibir parámetros, como cualquier otro proceso. Es decir, que la inclusión de esta palabra tiene un aspecto más clarificador, ya que en DIV1, ya se podía crear un proceso con el



mismo comportamiento, lo que pasa es que ahora podremos indicarlo de forma visible en el listado.

Ahora pasemos a ver las funciones relacionadas con este par de trucos, que son solamente dos.

- *int memory_free():* esta función devuelve un valor que nos permite saber la memoria libre que queda en el sistema.
- *int ignore_error(numero):* como parámetro de esta función deberemos indicar el número de error que queremos ignorar. Su uso es sencillo, pero como hemos comentando anteriormente, no es recomendable.

Funciones de color

Por último, pasamos a un apartado, que aunque no puede parecer importante, en algunos casos no será así. Antes de introducir los trucos que podemos conseguir trabajando con el color comentaremos un par de aspectos de DIV2, que podremos considerar como negativos.

La opción de "ignorar error" puede ser necesaria con los modos 8

El primero de ellos es el fallo de la función *roll_pallete()*. Esta función, que sirve para rotar la paleta, ha dejado de funcionar en DIV2. Este pequeño bug, es de carácter interno, por lo que no tiene solución, es decir, no se puede hacer nada para que dicha función vuelva a su estado normal. Aunque sí podemos conseguir un efecto parecido con las otras funciones. Aunque no podemos conocer el color que hay en una determinada posición, sí podemos designar una paleta establecida, que nosotros conozcamos totalmente, y moverla a mano, cambiando los valores, como veremos a continuación.

Ésa es la función que veremos ahora, ya que ahora existe la posibilidad de crear nuestra propia paleta desde cero. Esto, aunque parece muy simple, es muy potente, ya que al poder designar un color de la paleta con la función *set_color()*, podremos crear nuestra propia paleta, con todo lo que ello conlleva. Y uniendo esto al párrafo anterior, donde hablábamos de *roll_pallete()*, y la posibilidad de crear nuestro pro-

Cuadro 1. La lista de correo y el DIVCD

Aquí vamos a ver un par de temas relacionados con DIV. El primero es que la lista de correo ha cambiado de sitio. Si antes estaba en el servidor *onelist*, cuya página era *http://www.onelist.com*, ahora, han movido todas las listas a uno nuevo, llamado *egroups*, y cuya pagina es *http://www.egroups.com*. Por lo tanto, la lista pasaría a denominarse *canaldiv@egroups.com*.

Por cierto, últimamente la lista está muy revolucionada. Se tratan temas como la calidad de esta revista, ***DIVmanía***, el futuro de otra revista electrónica que muchos conocéis, y que se llama DIVnet, y otros temas. Pero un tema que ha resultado ser muy impactante es el siguiente. Una empresa, llamada Top Games, de origen argentino, ha sacado a la venta algunos juegos hechos con DIV, no se sabe a ciencia cierta si con el consentimiento de sus autores. En la lista, se ha optado por la siguiente solución: se ha decidido crear un CD, con todos los juegos de DIV posibles, cuyo nombre provisional es DIVCD. El proyecto está todavía en sus primeros pasos y esperamos que llegue a buen término. No es seguro pero, en principio, sería un CD con la mayor cantidad posible de juegos hechos con DIV gratuito, o lo más barato posible, y se distribuiría por distintas revistas o por alguna distribuidora.

Otro tema, también de interés es el DIVsite, un lugar en Internet que albergase todo el mundo de DIV. Esta pagina, o mejor podríamos llamarlo servidor, estaría creado y mantenido por los usuarios, y contendría revistas de electrónica, juegos, y todo lo que se ha generado, y lo que tiene que venir, sobre el mundo DIV. Desde estas líneas apoyamos dicha idea.

pio *roll_pallete*, vamos a ver un truco para conseguirlo. Podremos tener en una tabla, nuestra propia paleta preestablecida. Luego usando la función *set_color()*, la introduciremos dentro del programa, aunque sólo sea la parte que queremos rotar. Por último, para conseguir la rotación de colores, únicamente debemos cambiar el índice de la tabla. Es decir, iremos eligiendo un distinto inicio y fin, de selección de color, con esto conseguiremos que la paleta rote, como hacía la función *roll_pallete()*.

Además de poder crear una paleta única con todos los gráficos que tengamos, existe una nueva función en DIV2, cuya utilidad es muy obvia. Se trata de *force_pal()*, que nos permite tener una sola paleta para todo. Se acabó lo de convertir paletas y font de letras, de forma exterior a la programación. Ahora, usando esta función, mediante programación, conseguiremos que todos los gráficos y font, se adapten automáticamente a una paleta determinada. Esto nos quitará mucho trabajo, aunque debido al número de colores, mas de 16 millones, la adaptación no será siempre la deseada, simplemente, porque en la paleta que tengamos cargada no exista.

Por último, comentaremos una función cuya utilidad no ha encontrado el que escribe, aunque esto no signifique que no la tenga, simplemente, no se me ha dado ningún caso donde utilizarla. Se trata de *find_color()*, que se utiliza para buscar la posición de un color dentro de la paleta. Esta función, unida a *set_color()*, puede servir para cambiar dos colores determinados de posición.

Podemos tener en una tabla nuestra propia paleta de colores

Bueno, ahora pasemos a ver las funciones destinadas al color, de forma más detenida. La lista es la siguiente:

- *int set_color(color,r,g,b)*: esta función sirve para designar un color dentro de la paleta, es decir, crear un color propio. Como parámetros debemos indicar la posición de color dentro de la paleta, y los componentes RGB del color que queremos crear. Estos componentes RGB, son valores que definen el color. El componente R, que proviene del inglés, *Red*, designa la cantidad de color rojo que tendrá el que queremos crear. El componente G, *Green*, significa verde, y el componente B, *Blue*, significa azul. Juntando estos tres componentes, podremos conseguir cualquier color.
- *int find_color(r,g,b)*: esta función sirve para buscar un color determinado dentro de la paleta. Debemos utilizar como parámetros el color, dividido en componentes RGB, como hemos explicado en la función anterior. La función devuelve el color más cercano al indicado, y los componentes tendrán valores entre 0 y 63.
- *int force_pal(nombre archivo)*: por último, tenemos esta función que sirve para forzar al programa a utilizar solamente esta paleta. El entorno DIV convertirá automáticamente a dicha paleta todos los fondos o gráficos que carguemos en memoria. Debemos indicar como parámetro el nombre del archivo del cual usaremos la paleta. Para dejar de forzar la paleta, se usará como parámetro el valor 0, volviendo al funcionamiento normal. Es decir, los gráficos no se adaptarán a la paleta cargada en memoria, sino que utilizarán la propia.

Cuadro 2. Ediv y Fenix (El fenómeno DIV)

Se está creando por parte de usuarios otros programas para crear juegos. Éstos están basados en la esencia de DIV, se llaman Fenix y EDIV. Los dos entornos de programación son gratuitos, por lo que cualquier usuario puede descargarlos desde Internet para su posterior prueba. Aunque para la gente que no tiene conexión a la Red, desde estas líneas, se intentará hacer llegar estos productos a esta revista.

Y es que, realmente, DIV ha roto esquemas. El número de usuarios apuntados a sus listas de correo es enorme. En cualquier ida, si uno se pasa por el canal a eso de la media noche, se puede encontrar fácilmente a 10 ó 20 personas. El número de páginas que tratan sobre DIV es también enorme. Y esto es sólo la gente que tiene conexión a Internet, es decir, un tanto por ciento del número real de usuarios que, seguramente, será muy grande.

Por eso, la gente no quiere quedarse en DIV2 e intenta, por su cuenta, realizar productos parecidos, aunque sean gratuitos. Esto es así, porque la idea de DIV es genial, ya que la rapidez con que se puede hacer un videojuego con este producto es muy grande. Además de su sencillez de uso, y el que dentro de la misma herramienta se cuenta con utilidades de programación, de edición de gráficos, e incluso de manejo de sonido.

Bueno, sólo desear suerte a la gente que está trabajando en estos proyectos. Y, desde aquí esperamos que sus productos lleguen a buen fin. Consiguiendo, por lo tanto, terminarlos y darles salida, de la forma que sea, para el disfrute de todos los usuarios que así lo quieran.

Rutinas de red

Para empezar con esta sección habría que dejar muy claro que el que escribe no ha hecho pruebas con las rutinas de red, por lo cual, los consejos que se pueden dar son mínimos. Hasta aquí han llegado rumores de que tales rutinas funcionan, pero no se ha comprobado nada todavía. La mayor parte de la gente que lo ha intentado, ha tenido serios problemas para hacerlas funcionar. Es verdad, que dichas rutinas tienen muchos errores o bugs, y el número de pruebas que se hicieron con ellas antes de salir al mercado fue mínimo.

Esto pasa igual en el modo 8, que también queda por experimentar, aunque se han hecho muchas pruebas. Al contrario, las rutinas de red, más difíciles de probar, por el simple hecho de que hay que disponer de una red IPX y, por lo tanto, de más de un ordenador para poder hacer pruebas. Por eso, en modo 8, se han encontrado miles de trucos como, por ejemplo, no juntar más de dos sectores en un punto, intentar no unir paredes en la misma posición, no usar muchos vértices, etc. Sin embargo, esto no ocurre con las rutinas de red, donde las pruebas son mínimas y poco satisfactorias.

Por esta razón, para no crear un castillo en el aire explicando algo que realmente no se sabe a ciencia cierta, lo dejamos en este punto, pidiendo a cualquier lector, que si ha hecho pruebas satisfactorias con las rutinas de red, utilizando cualquier tipo de truco, se lo haga saber a la lista de DIV, a esta revista o, simplemente, al que escribe este artículo.

Despedida y cierre

Con este artículo se cierra la serie. Hemos ido recorriendo todas las novedades que incorpora DIV2, viendo algunos trucos sobre cada una de ellas. Espero que todos estos trucos os sean útiles o que, por lo menos, estos artículos os hayan servido para repasar todas las mejoras de DIV2. Ya sabéis que si tenéis alguna duda, siempre la podéis hacer llegar a la revista, o si lo preferís, a un servidor, cuya dirección de correo electrónico es: *tizo@100mbps.es*. Nada más, como cada mes, sólo queda decir lo siguiente: "Que os DIVirtáis programando".

Antonio Marchal (Tizo)

Mapeados Tiles en vista lateral

Infinidad de fases para tu juego de plataformas

Conectando con el anterior artículo de arcades y plataformas, dedicamos el espacio en esta ocasión a la creación e introducción al mundo de los tiles y los mapeados tiles en 2D de vista lateral. Un recurso que nos va a permitir crear escenarios diferentes en muy poco tiempo.

El mundo de los tiles siempre ha resultado una tediosa tarea y quizás un gran inconveniente para su propagación efectiva. Intentaremos pues, y debido a que muchos son los grafistas y programadores que desconocen lo que es este apasionante mundo, dar las notas iniciales para adentrarnos en él.

Los tiles han sido siempre un gran recurso en la programación de videojuegos, que nos permite crear grandes escenarios con fragmentos de lo que podríamos llamar un "mosaico gráfico".

Para la creación de un tile tenemos que saber el tipo de rejilla a utilizar

Un tile no es ni más ni menos que un pequeño fragmento de forma rectangular que, unido a una serie escasa de los mismos, tiene la virtud de crear escenarios de gran amplitud. Muestra de ello, son juegos como Super Mario World del que os enseñamos una muestra de sus tiles (ver foto 1). Lo que hace característico a un tile es su forma especialmente diseñada, como un patrón que puede ser ampliado hasta el infinito sin la sensación de repetición y su limitación se encuentra precisamente en la imaginación y astucia de su creador. Pero aún es mucho más que eso pues los años han demostrado que estos recursos tenían todavía mucho que ofrecer, y muestra de ello son los juegos de estrategia de última generación.

¿Cómo podemos empezar a crear tiles?

Para la creación de un tile, lo primero que tenemos que saber es el tipo de rejilla que vamos a utilizar. La rejilla (ver foto 2) será el espacio imaginario que nos servirá para encajar los tiles unos con otros y crear el escenario final. Es por ello imprescindible que todos los tiles sean, "en principio" del mismo tamaño. Sólo en principio porque aunque el tamaño real de los tiles sí sea efectivamente igual al del resto, con el uso de las transparencias (en DIV, el color 0 de la paleta cargada) podemos crear tiles no necesariamente cuadrados. Es también muy aconsejable, con vistas a la programación posterior que los tiles sean proporcionales a la resolución de la pantalla, es decir, a una resolución de 640 x 480 en la que sólo vamos a utilizar un scroll lateral en 2D (con lo que sólo debemos tener en cuenta los 640 píxeles de ancho de la pantalla) que sus tiles sean un divisor exacto del mismo, en este caso, anchos de 64 (64 x 10=640 píxeles) 32 (32 x 20=640 píxeles) o 16 (16 x 4=640 píxeles), con vistas a la programación de tileados con scroll automático que veremos en el próximo número.

El siguiente paso es imaginar el tile. Los tiles suelen representar fragmentos de objetos, como

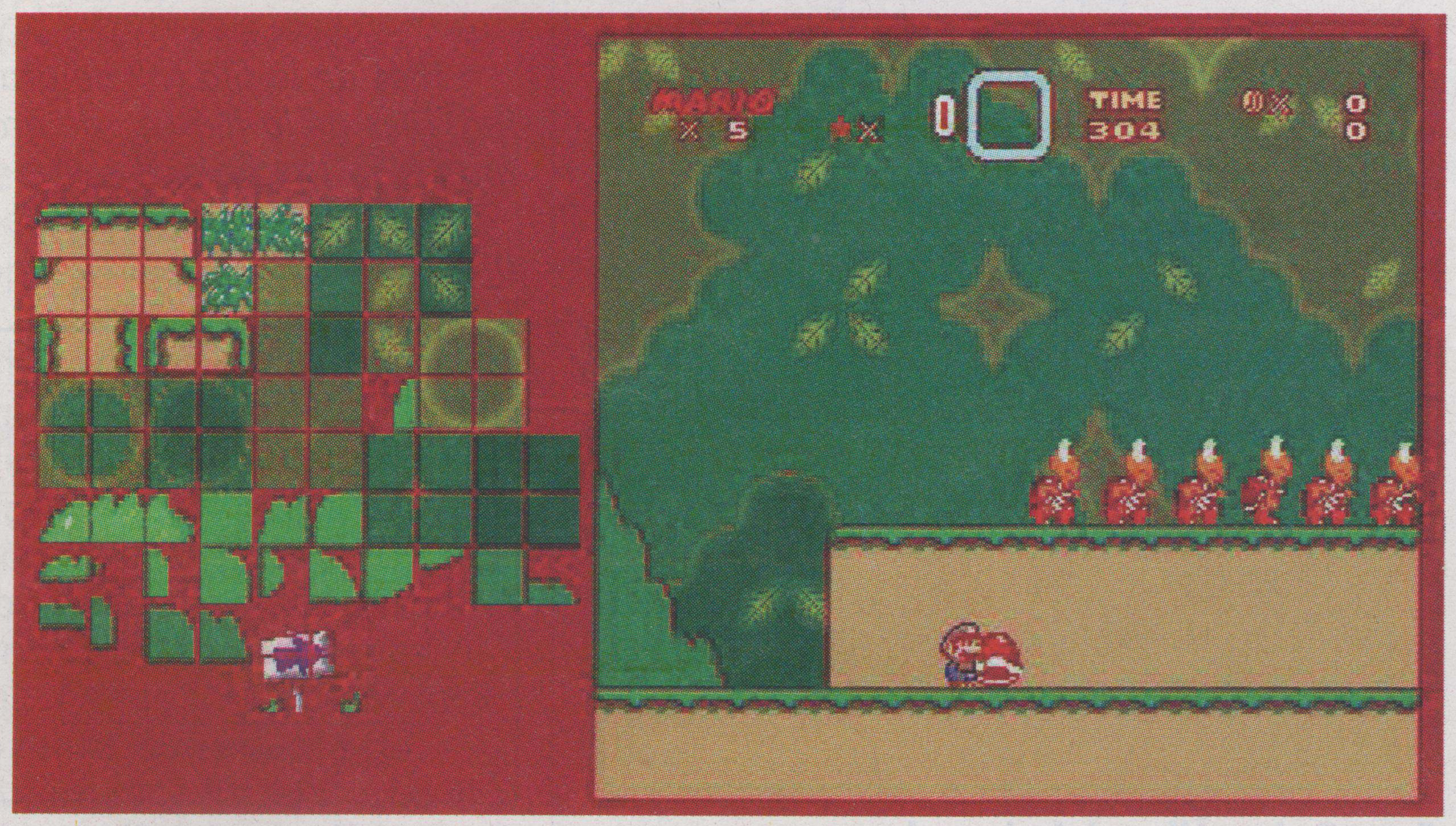


Foto 1. Los Tiles de Super Mario World.

pio *roll_pallete*, vamos a ver un truco para conseguirlo. Podremos tener en una tabla, nuestra propia paleta preestablecida. Luego usando la función *set_color()*, la introduciremos dentro del programa, aunque sólo sea la parte que queremos rotar. Por último, para conseguir la rotación de colores, únicamente debemos cambiar el índice de la tabla. Es decir, iremos eligiendo un distinto inicio y fin, de selección de color, con esto conseguiremos que la paleta rote, como hacía la función *roll_pallete()*.

Además de poder crear una paleta única con todos los gráficos que tengamos, existe una nueva función en DIV2, cuya utilidad es muy obvia. Se trata de *force_pal()*, que nos permite tener una sola paleta para todo. Se acabó lo de convertir paletas y font de letras, de forma exterior a la programación. Ahora, usando esta función, mediante programación, conseguiremos que todos los gráficos y font, se adapten automáticamente a una paleta determinada. Esto nos quitará mucho trabajo, aunque debido al número de colores, mas de 16 millones, la adaptación no será siempre la deseada, simplemente, porque en la paleta que tengamos cargada no exista.

Por último, comentaremos una función cuya utilidad no ha encontrado el que escribe, aunque esto no signifique que no la tenga, simplemente, no se me ha dado ningún caso donde utilizarla. Se trata de *find_color()*, que se utiliza para buscar la posición de un color dentro de la paleta. Esta función, unida a *set_color()*, puede servir para cambiar dos colores determinados de posición.

Podemos tener en una tabla nuestra propia paleta de colores

Bueno, ahora pasemos a ver las funciones destinadas al color, de forma más detenida. La lista es la siguiente:

– *int set_color(color,r,g,b):* esta función sirve para designar un color dentro de la paleta, es decir, crear un color propio. Como parámetros debemos indicar la posición de color dentro de la paleta, y los componentes RGB del color que queremos crear. Estos componentes RGB, son valores que definen el color. El componente R, que proviene del inglés, *Red*, designa la cantidad de color rojo que tendrá el que queremos crear. El componente G, *Green*, significa verde, y el componente B, *Blue*, significa azul. Juntando estos tres componentes, podremos conseguir cualquier color.

– *int find_color(r,g,b):* esta función sirve para buscar un color determinado dentro de la paleta. Debemos utilizar como parámetros el color, dividido en componentes RGB, como hemos explicado en la función anterior. La función devuelve el color más cercano al indicado, y los componentes tendrán valores entre 0 y 63.

– *int force_pal(nombre archivo):* por último, tenemos esta función que sirve para forzar al programa a utilizar solamente esta paleta. El entorno DIV convertirá automáticamente a dicha paleta todos los fondos o gráficos que carguemos en memoria. Debemos indicar como parámetro el nombre del archivo del cual usaremos la paleta. Para dejar de forzar la paleta, se usará como parámetro el valor 0, volviendo al funcionamiento normal. Es decir, los gráficos no se adaptarán a la paleta cargada en memoria, sino que utilizarán la propia.

Cuadro 2. Ediv y Fenix (El fenómeno DIV)

Se está creando por parte de usuarios otros programas para crear juegos. Éstos están basados en la esencia de DIV, se llaman Fenix y EDIV. Los dos entornos de programación son gratuitos, por lo que cualquier usuario puede descargarlos desde Internet para su posterior prueba. Aunque para la gente que no tiene conexión a la Red, desde estas líneas, se intentará hacer llegar estos productos a esta revista.

Y es que, realmente, DIV ha roto esquemas. El número de usuarios apuntados a sus listas de correo es enorme. En cualquier ida, si uno se pasa por el canal a eso de la media noche, se puede encontrar fácilmente a 10 ó 20 personas. El número de páginas que tratan sobre DIV es también enorme. Y esto es sólo la gente que tiene conexión a Internet, es decir, un tanto por ciento del número real de usuarios que, seguramente, será muy grande.

Por eso, la gente no quiere quedarse en DIV2 e intenta, por su cuenta, realizar productos parecidos, aunque sean gratuitos. Esto es así, porque la idea de DIV es genial, ya que la rapidez con que se puede hacer un videojuego con este producto es muy grande. Además de su sencillez de uso, y el que dentro de la misma herramienta se cuenta con utilidades de programación, de edición de gráficos, e incluso de manejo de sonido.

Bueno, sólo desear suerte a la gente que está trabajando en estos proyectos. Y, desde aquí esperamos que sus productos lleguen a buen fin. Consiguiendo, por lo tanto, terminarlos y darles salida, de la forma que sea, para el disfrute de todos los usuarios que así lo quieran.

Rutinas de red

Para empezar con esta sección habría que dejar muy claro que el que escribe no ha hecho pruebas con las rutinas de red, por lo cual, los consejos que se pueden dar son mínimos. Hasta aquí han llegado rumores de que tales rutinas funcionan, pero no se ha comprobado nada todavía. La mayor parte de la gente que lo ha intentado, ha tenido serios problemas para hacerlas funcionar. Es verdad, que dichas rutinas tienen muchos errores o bugs, y el número de pruebas que se hicieron con ellas antes de salir al mercado fue mínimo.

Esto pasa igual en el modo 8, que también queda por experimentar, aunque se han hecho muchas pruebas. Al contrario, las rutinas de red, más difíciles de probar, por el simple hecho de que hay que disponer de una red IPX y, por lo tanto, de más de un ordenador para poder hacer pruebas. Por eso, en modo 8, se han encontrado miles de trucos como, por ejemplo, no juntar más de dos sectores en un punto, intentar no unir paredes en la misma posición, no usar muchos vértices, etc. Sin embargo, esto no ocurre con las rutinas de red, donde las pruebas son mínimas y poco satisfactorias.

Por esta razón, para no crear un castillo en el aire explicando algo que realmente no se sabe a ciencia cierta, lo dejamos en este punto, pidiendo a cualquier lector, que si ha hecho pruebas satisfactorias con las rutinas de red, utilizando cualquier tipo de truco, se lo haga saber a la lista de DIV, a esta revista o, simplemente, al que escribe este artículo.

Despedida y cierre

Con este artículo se cierra la serie. Hemos ido recorriendo todas las novedades que incorpora DIV2, viendo algunos trucos sobre cada una de ellas. Espero que todos estos trucos os sean útiles o que, por lo menos, estos artículos os hayan servido para repasar todas las mejoras de DIV2. Ya sabéis que si tenéis alguna duda, siempre la podéis hacer llegar a la revista, o si lo preferís, a un servidor, cuya dirección de correo electrónico es: *tizo@100mbps.es.* Nada más, como cada mes, sólo queda decir lo siguiente: "Que os DIVirtáis programando".

Antonio Marchal (Tizo)

puede ser el terreno. Para crear un terreno no exclusivamente homogéneo podemos crearlo de forma sencilla en tres tiles, poniendo especial atención en el tile que va en el centro puesto que será el que vamos a multiplicar para crear espacios de terreno más o menos amplios.

Cuando fabricamos un tile, debemos tener en cuenta su posición en la pantalla y su función, más concretamente, cuáles de sus lados van a ser compartidos con otros tiles, que deberán tener una forma determinada para que conecten con el tile con el que ha de unirse. En el caso del terreno del que hablábamos antes, la parte final izquierda de la misma debe tener su lado derecho pensado de tal manera que la parte central pueda continuar sus mismas líneas o formas sin que se note que son tiles independientes y de igual forma si va a tener partes inferiores; a la parte central y la misma tarea para ambos lados. Y ésa es precisamente la tarea más complicada de los tiles, para la que hay que tener quizás un poco más de habilidad, especialmente indicado para aquellas personas que dominan el dibujo píxel a píxel. Pero nada mejor que la práctica para mejorar la técnica.

La rejilla nos indica qué tile va en cada posición y en cada momento

Recordad que aunque los tiles suelen ser pequeños, si deseáis hacer objetos más grandes podéis utilizar varios tiles para un solo objeto, para lo que no tienen que ser en absoluto cíclicos.

Programación

Pasemos a la práctica. En esta primera parte, de los dos artículos que explicarán con detalles la programación de los mapeados tiles en 2D con scroll lateral, vamos a aprender a traspasar un mapa creado a partir de tiles a una tabla de datos con el orden de los tiles y desde esa tabla de tiles al mapa del nuestro juego.

Hay que recordar en todo momento que entre las muchas virtudes de usar los mapeados tiles

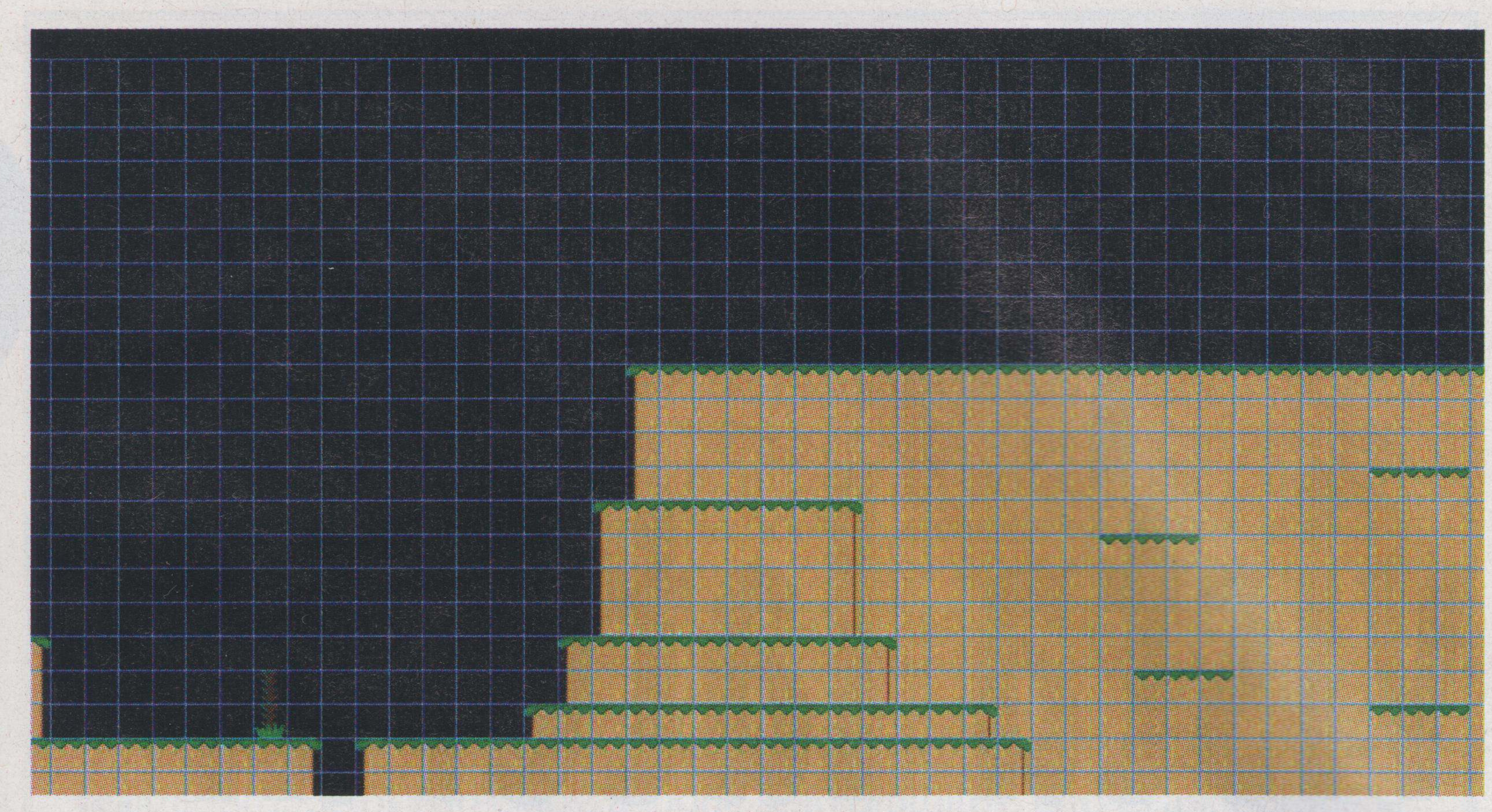


Foto 2. Los mapeados tiles se disponen en una rejilla imaginaria.

se encuentra la del ahorro de memoria y que, por tanto, vamos a intentar que nuestro juego use la memoria imprescindible, bien para dedicarla a otros menesteres (personajes y enemigos de gran tamaño, animaciones de los escenarios, o dotar al movimiento de una gran velocidad) o bien para que el programa pueda ser utilizado en ordenadores de gama baja.

Una vez tengamos los tiles preparados, creamos un mapa del tamaño que queramos para la fase y los encajamos creando el escenario en la forma que deseemos, dividimos el mapa con líneas según sean el tamaño de los tiles para conseguir una rejilla que nos permita saber qué tile va en cada posición en cada momento. Con esta rejilla, apuntamos los valores en una tabla bidimensional (en realidad unidimensional), anotando los que el tile de esa posición tenga otorgado en el archivo fpg donde estén almacenados, es decir, si el tile que vamos a pasar a la tabla tiene asignado el número 3 en el archivo fpg donde está guardado, entonces ése es el número que hay que insertar. Debería quedar como la que se muestra a continuación:

```
    Tiles de la fila (nº de graph en
fpg)Tiles de la fila (n_ de graph en
fpg)
    0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,              _ fila 0
    0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,              _ fila 1
    0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,              _ fila 2
    1,2,2,2,2,3,0,0,1,2,2,
2,2,2,2,2,3,0,              _ fila 3
    5,4,4,4,4,6,0,0,5,4,4,4
,4,4,4,4,6,0,               _ fila 4
```

Tabla que traspasaremos a DIV, como se señalaba, de forma unidimensional. En realidad es posible, incluso visualmente más correcto, dividir esta misma tabla de la siguiente forma, y no en forma lineal que sería de difícil visión.

```
Byte tiles[89] =
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  1,2,2,2,2,3,0,0,1,2,2,2,2,2,2,2,3,0,
  5,4,4,4,4,6,0,0,5,4,4,4,4,4,4,4,6,0;
//recuerde siempre el ";" al final.
```

Para acceder de forma bidimensional a esta tabla debemos encadenar dos bucles, uno para las filas y otros para las columnas o tiles de esa fila. Teniendo en cuenta que en este ejemplo cada fila contiene 18 tiles, para acceder a las filas posteriores a la 0, tendremos que multiplicar por 18 el número de la fila del tile a la que deseemos acceder, sumándole el número de la posición (o columna)del tile dentro de esa misma fila. Lo que traspasado a un bucle quedaría de la siguiente forma:

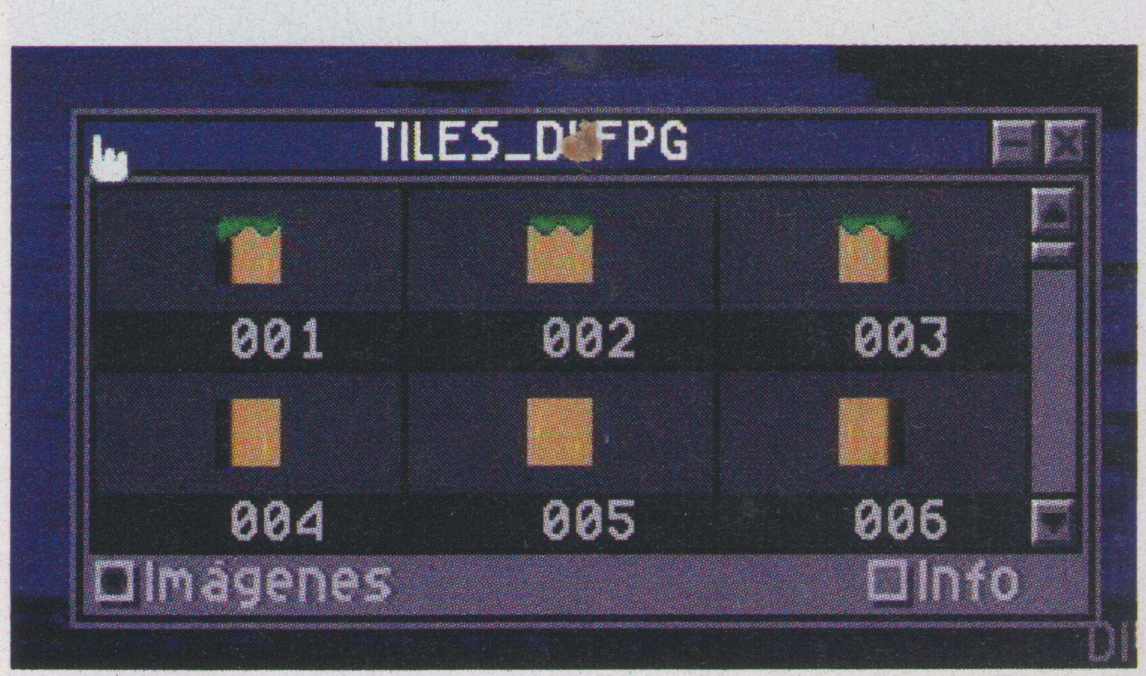


Foto 3. Los tiles del ejemplo.

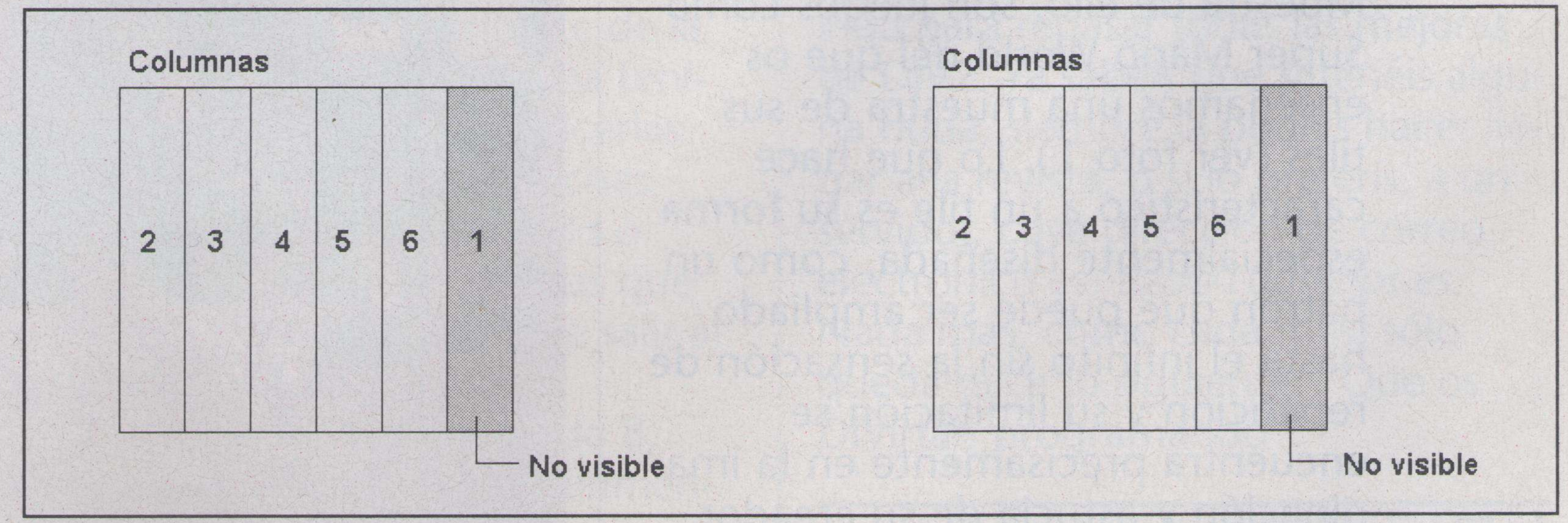


Foto 4. Figura 1 y Figura 2.

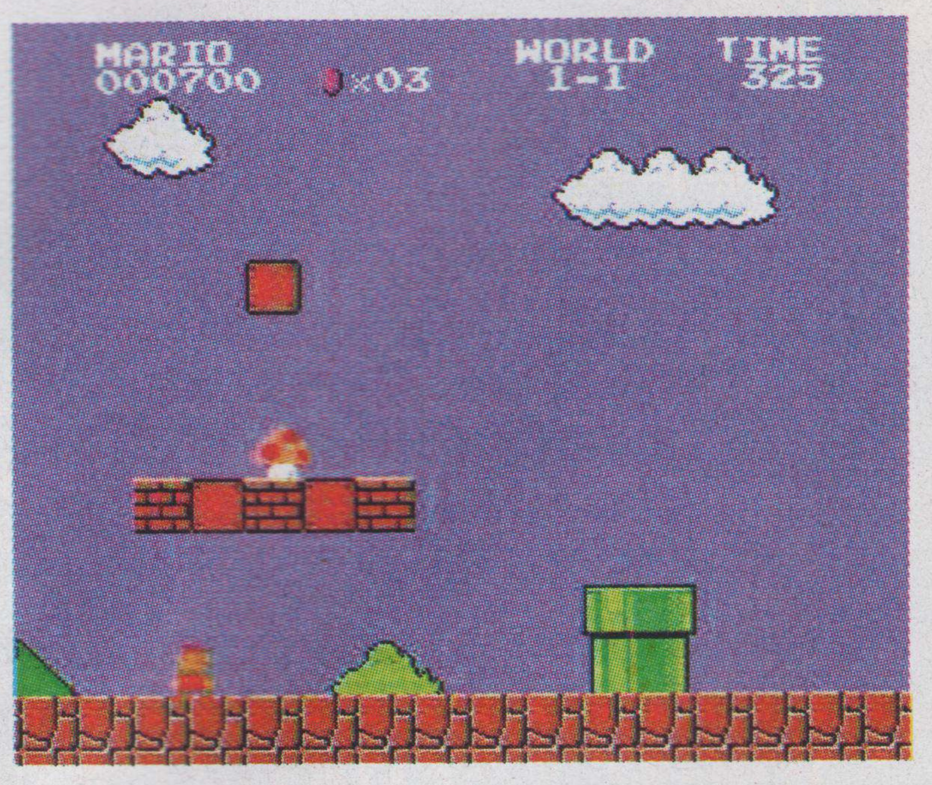


Foto 5. Super Mario Bros, todo un clásico de las plataformas.

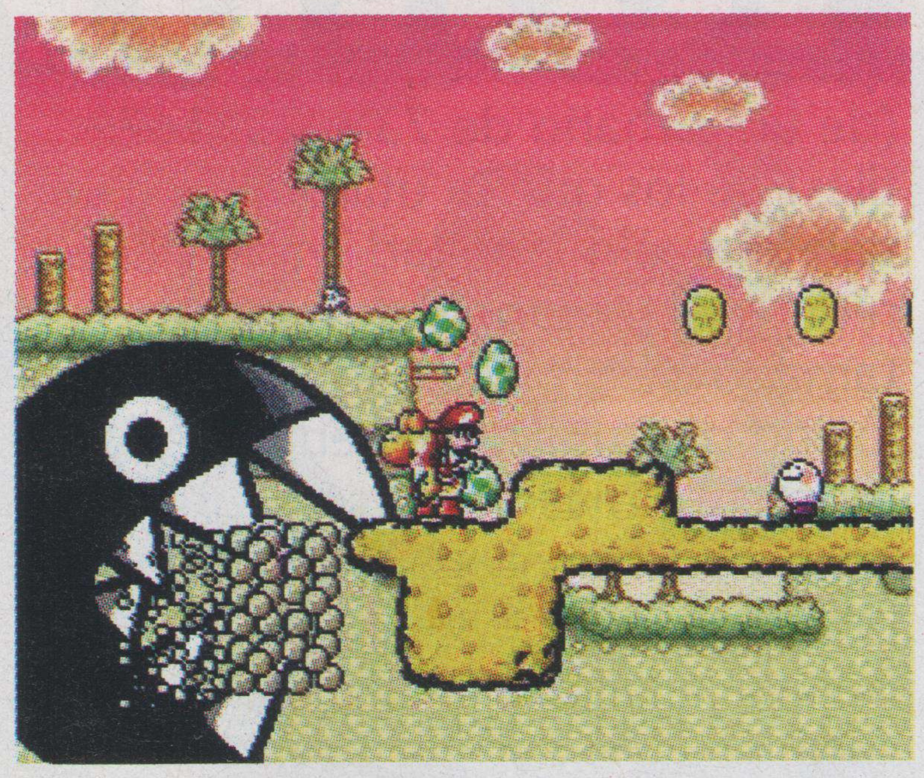


Foto 6. El género de las plataformas cada vez más complejo.

```
From (fila=0 to 4)
From (columna= 0 to 17)
  [...]
End
End
```

Lo que conseguiremos con este bucle es un acceso lineal a los tiles. En cada fila, el bucle ejecuta los 18 pasos (0-17) para la lectura de las respectivas columnas correspondientes a tiles de la fila en cuestión. Por lo que sólo nos queda acceder a los tiles y encajarlos en nuestro mapa con *map_put* que nos permite introducir un gráfico en otro. Tan sólo un problema, recordad que *map_put* encaja los gráficos dentro de otros pero siempre dentro del mismo fichero fpg. Esto quiere decir que si creáis el mapa con *new_map*, no os será posible encajar tiles que no sean correspondientes al fichero 0, que es el fichero por defecto de los mapas creados en tiempo de ejecución. Teniendo en cuenta esto, sólo no quedan dos soluciones:

A) Que el fichero 0 que carguemos siempre sea el correspondientes a los tiles.
B) Que dentro de cada fichero incluyamos un mapa del tamaño de la fase de color transparente.

Veamos, ahora sí, cómo sería la función que nos permitirá encajar los tiles en el mapa:

```
From (fila=0 to 4)
From (columna=0 to 17)

 I= tiles[(fila*18)+columna];
 If (i>0)

   Map_put(0, nº_map_destino,tiles[(fila*18)+columna],
        columna*acho_tile,fila*alto_tile);
   End
 End
 End
```

Como se puede observar, en el bucle accedemos al *n* de gráfico que devuelve la posición del tile consultado y sólo accedemos a introducirlo en el mapa si el *graph* devuelto no es 0. Igualmente, para situarlos en el mapa, utilizamos una sencilla fórmula que consiste en multiplicar la columna por el ancho del tile, que nos dará su posición horizontal, y la fila por el alto del tile, que nos dará la posición vertical del mismo. Con esto, ya sólo nos queda crear el scroll correspondiente para mover horizontalmente el mapa, en caso de que sea mayor que la resolución que deseemos.

Para realizar un scroll necesitamos al menos un mapa, que será el de la fase, aunque se pueden poner algún otro que haga de fondo para crear un scroll parallax que podría dar un toque de calidad a nuestro juego, y también un proceso, en este caso el principal que activaremos como la scroll camera, con *scroll.camera=id.* Quedando algo así:

```
START_SCROLL(0,fpg_tiles,
9,0,0,0);
// en este caso dejamos en blanco el segundo mapa y no permitimos que el mapa se repita de forma cíclica
Scroll.camera=id;
```

Y para controlar el scroll simplemente:

```
If (key(_left))
  If (x>0) x-=4; END
End
If (key(_right))
  If (x<1500) x+= 4 END
END
```

Conceptos básicos de un mapper

Un mapper es un programa que permite hacer de forma automática las tablas de tiles a partir de un diseño que nosotros mismos hayamos definido para un determinado escenario. Es bastante útil y práctico porque realizar estas actividades "a mano" puede resultar bastante monótono. Las utilidades más características de los mismos incluyen una rejilla que adapta de forma automática los tiles que el usuario ha seleccionado para una casilla determinada de esa rejilla. Para evitar que el usuario coloque un tile en una zona que no es exactamente la correcta podemos mirar las coordenadas del ratón en el momento de la pulsación y colocarlo en el cuadrante que corresponda a esa situación del ratón. Por ejemplo, si se trata de una rejilla de tiles de 20 x 20 podemos comprobar en qué cuadrante está el ratón dividiendo por esos mismos valores las coordenadas de x e y: *x/20=columna* y *y/20=fila.* Otro aspecto importante es la posibilidad de automatizar totalmente la creación de escenarios, es decir, no sólo presentar los tiles la parte derecha de la pantalla para que el usuario los pulse y los ponga en la rejilla sino poner tan sólo iconos y usar el método de pulsar y arrastrar el ratón para que, por ejemplo, una sección de terreno se ponga de forma automática correctamente, es decir, de todas sus partes (parte izquierda, central y derecha, terreno debajo de terreno, etc) con lo que podremos incluso dar la posibilidad al jugador de crear él mismo sus propias fases sin ningún tipo de problema. En el próximo número se mostrará un sencillo ejemplo de cómo crear un mapeador de tiles y comentaremos las partes más importantes del código.

Un mapper es un programa que permite hacer tablas de tiles

Un mapeado tile con scroll avanzado con scroll automático

Decíamos anteriormente que una de las grandes virtudes que tienen los mapeados tiles es su bajo uso de los recursos del sistema. Este no es el caso del ejemplo que damos en este número pues un mapa de 1500 x 480 píxeles puede pesar bastante en memoria. Sólo es un ejemplo para mostrar cómo se empiezan a manejar los mapeados tiles y su programación. Una vez terminado este paso, debemos idear un sistema que de



Foto 7. Los Tiles, un recurso acogido en mucho géneros de videojuegos.

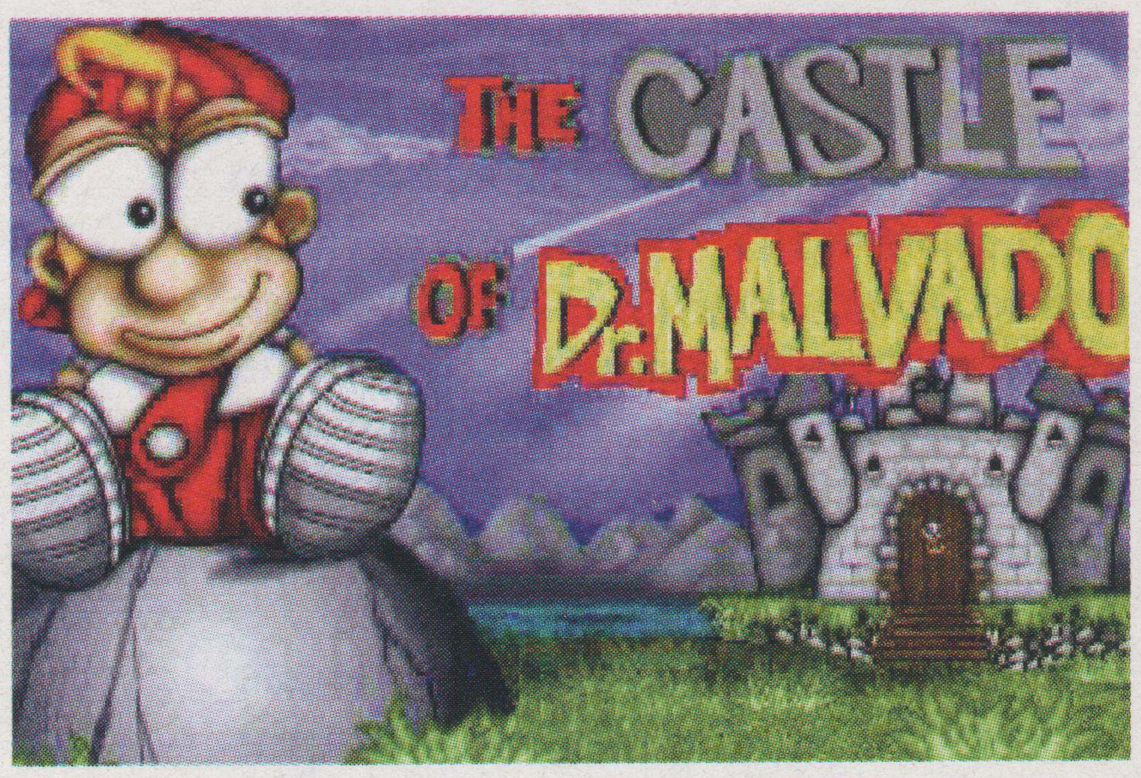


Foto 8. El inolvidable Dr. Malvado, el mejor plataformas de DIV.

forma automática muestre en la pantalla sólo lo que estamos viendo exclusivamente. Pensemos por un momento en los programas de estrategia que en su gran mayoría usan igualmente técnicas de mapeado de tiles. Si vemos el caso de Warcraft, las fases en extensión suelen ser bastante amplias superando los mapas incluso los 5000 x 5000 píxeles e inumerables personajes interactuando de forma continua en tiempo real. ¿Cómo es posible algo semejante de forma fluida? Pues la única respuesta es no mostrando lo que no se ve en ese momento, es decir, mostrar sólo aquellos gráficos que exclusivamente se están visualizando, suspendiendo la parte gráfica de los mismos que no lo están. Algo que traducido a los mapas de tiles se traduce en mostrar sólo aquellas fracciones de tiles que se están visualizando e ir actualizando la parte correspondiente a las regiones que van a ser vistas inmediatamente. En DIV, el único método posible que permite semejante cosa es la interacción entre un scroll y *new_map* de la forma que se va a describir teóricamente a continuación y prácticamente con un ejemplo en el siguiente número de acción y plataformas.

Los tiles suelen ser pequeños, para un objeto grande se usan varios tiles

Teniendo como base la misma tabla que usamos para el ejemplo que viene con el CD de la revista (de un tamaño de 73 x 23=1679 tiles) de los que se utilizan en pantalla tan sólo la mitad, vamos a reducir el tamaño del mapa al tamaño de un mapa igual a la resolución que utilicemos en ese momento más un incremento en su ancho y alto correspondiente al tamaño también de ancho y alto de tipo de tiles que usemos, en este caso de 20 x 20, por lo que en una resolución de 640 x 480 deberíamos utilizar un mapa de 660 x 500 y lo ponemos como mapa principal de un scroll cíclico (horizontalmente, o también verticalmente si vamos a esa posibilidad). Procedemos de la misma forma que se explicó y ponemos los tiles correspondientes a toda la pantalla de visión y la parte no visible de actualización que le añadimos. Precisamente en esa región de 20 píxeles que no se ve, será la parte que iremos actualizando según vayamos necesitándola, esto es, que cuando el jugador mueva el mapa hacia la derecha, el scroll moverá las regiones visibles quedando no visibles las primeras que aparecieron y entrando en las regiones visibles la parte que nosotros dejábamos para actualizar. Veámoslo de forma gráfica con un ejemplo.

Imaginemos una rejilla de tan sólo seis columnas (ver fig.1), donde la última no es visible y para poder actualizar los tiles sin que el usuario vea ningún cambio. Cuando el jugador mueve el mapa y la región 6 que antes no era visible se muestra en su totalidad en la pantalla, la región 1 queda totalmente oculta en la parte que ocupaba la 6, por lo que nuestro algoritmo deberá actualizar aquellas zonas del mapa que no estén visibles en ese momento.

Otra cuestión muy importante a tener en cuenta a la hora de crear un scroll automático de tiles avanzado es los mapas de durezas necesarios para que los personajes y enemigos puedan adaptarse a los mapas creados. Debido a que sería igualmente un inconveniente, en lo que a requerimiento de memoria supondría, tener cargado en memoria un mapa sólo para medir las durezas de nuestro mapeado de tiles. Para resolver esta cuestión, lo único que debemos hacer es un tileado paralelo al nuestro que, incluso aprovechando la misma rejilla y con los mismos números que anteriormente tenían los tiles, crear copias de esos mismos tiles que usábamos, pasarlo a escala de grises y pintar sobre ellos las zonas que impiden pasar o que permiten que el personaje pueda pasar por encima. En aquellos tiles que sean indiferentes para el mapa de durezas, es decir, que no tienen ningún efecto con los personajes o enemigos, simplemente, podemos dejarlos como transparentes.

Dejaremos el resto para el siguiente número que lo dedicaremos en exclusiva a comentar el código del ejemplo, en el que se mostrará un ejemplo de scroll tile avanzado, un mapper para mostrar cómo automatizar las tareas. Mucha teoría y muy poca práctica en este número, en el siguiente tendremos todo lo contrario. Hasta entonces podéis mandar cualquier tipo de duda o sugerencia en relación con la sección a *migens@teleline.es* o en la lista del canal en egroups (*www.egroups.com*) en *canaldiv@egroups.com*.

José Antonio Migens Gómez (ViTAL).
migens@teleline.es



Ejemplo de juego con scroll lateral.

Dudas de los lectores

Nos llega la consulta de un usuario preocupado por un problema que ha encontrado con DIV, provocado por el intérprete de la sintaxis de DIV que se queda colgado cuando se declara cualquier tipo de variable con el único texto "con". No aparece el problema si está acompañado de cualquier otro tipo de carácter.

Respuesta

Curioso problema el que nos comentas. Como al parecer el problema sólo ocurren con las variables strings con el texto "con" para utilizarlo en las comparaciones en cualquier texto o cualquier otro tipo de acción podemos añadirle en la declaración de la variable cualquier tipo de carácter al final y dentro del programa, al inicio o en el mismo momento que te haga falta eliminar el último carácter de la variable. Esto es posible debido a que parece que el problema se encuentra en el momento de interpretar la validez del código y no en el tiempo de ejecución.


Millennium
Chess
Un juego ancestral de estrategia
El mejor ajedrez virtual
Emula a los maestros del tablero
Desarrolla tu inteligencia
Perfecciona tu estilo ajedrecístico
Sólo
18,00 €
PC CD rom
2.995
Ptas
De venta en quioscos, grandes superficies
y tiendas especializadas
Disfruta del milenario juego del ajedrez desde otra dimensión. "Millennium Chess" es un simulador de ajedrez con el que podrás desafiar a familiares y amigos o al propio ordenador. Incluye manual con amplia información
PC CD rom
COMPATIBLE
WINDOWS 98
Software en castellano
Digital Dreams MULTIMEDIA
Digital Dreams Multimedia
C/ Alfonso Gómez, 42, nave 1-1-2
28037 Madrid (España)
Fax: +34 91 304 17 97
www.ddmultimedia.com
Para más información,
llamar al teléfono +34 91 304 06 22

Trabajando con cadenas

Usando cadenas de texto

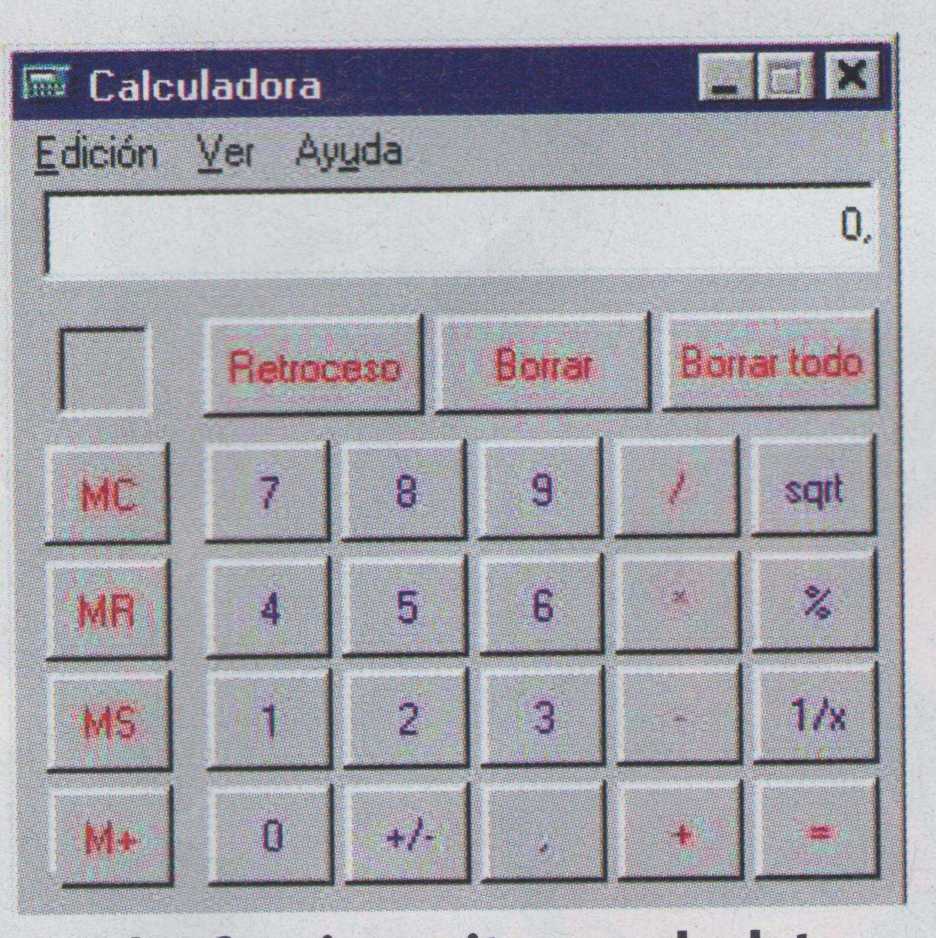
Un aspecto muy importante de un juego es la interactividad con el usuario. Es por ello que conviene diversificar las formas de comunicación con el jugador. Encontraremos en las cadenas una herramienta muy útil para construir interfaces hombre-máquina más eficaces.

Hasta ahora, tan sólo nos hemos comunicado con nuestros juegos mediante el teclado, joystick y ratón, para realizar desplazamientos de procesos. Pero en muchas ocasiones, desearemos utilizar el teclado para escribir texto y poder procesarlo posteriormente. Un claro ejemplo es una tabla de récords, en la que debemos introducir nuestras iniciales. En este número veremos cómo podemos captar y operar con cadenas de texto, pero para ello, en primer lugar, debemos conocer la nueva estructura de declaración de datos en DIV 2: los tipos de datos.

Tipos de datos

¿Qué es un tipo de datos? No es más que la forma de almacenamiento de los distintos valores de un programa. En la primera versión de DIV, podíamos declarar variables globales, locales y privadas, sin atender al tipo de dato que representaban, ya que todos se almacenaban de la misma forma. Esto también se puede realizar en DIV 2, si bien es recomendable utilizar la nueva capacidad de que dispone. Con los tipos de datos, podemos indicar la forma de almacenarlos en nuestro juego. Muchos de vosotros pensareis que es una carga más a la hora de programar. Es cierto, pero nos proporciona una mayor robustez en el código producido, ya que se pueden detectar con mayor facilidad errores como la asignación de un valor de cadena a una variable numérica.

Un tipo de datos es la forma de almacenamiento de los valores de un programa



Con las funciones itoa y calculate podemos usar DIV 2 como si de una calculadora se tratara.

En DIV 2 disponemos de varios tipos de datos, que describimos a continuación:

- **INT:** es el valor por defecto, es decir, el que se toma si no se indica ningún tipo de dato. Era el usado por la primera versión de DIV para todas sus variables numéricas. Su rango de valores oscila entre –2147483648 y +2147483647.
- **WORD:** la variable tiene un rango de valores entre 65535 y 0. No admite números negativos, con lo que para usarlos, debemos utilizar el tipo de datos INT.
- **BYTE:** toma datos entre 255 y 0.
- **STRING:** es un tipo específico de tablas que nos permitirá almacenar una cadena de caracteres, es decir, una sucesión de letras.

Todas las funciones del lenguaje han sido modificadas para aceptar como parámetros estos nuevos tipos de datos, si bien, se sigue manteniendo compatibilidad al 100% con la versión anterior.

Además de estos tipos de datos, disponemos de otras estructuras que nos permiten acceder a los datos de una forma mucho más cómoda y segura. Estas son las tablas y las estructuras.

Las tablas no son más que una lista de valores a los que se puede acceder mediante su posición en la misma, es decir, si tenemos una lista de valores BYTE como la que sigue:

2,4,6,8,7,1,2,10

Para acceder al valor 6, debemos indicar su posición, en este caso la posición 2 (comenzando por 0). Gráficamente, una tabla puede representarse de la siguiente forma:

Posición	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*
Valor	*2*	*4*	*6*	*8*	*7*	*1*	*2*	*10*

Para crear una tabla debemos seguir la siguiente sintaxis:

<tipo de dato> nombre tabla [tamaño de la tabla];

Para asignar a cada posición un dato (siempre del tipo de dato indicado en la declaración), debemos seguir la siguiente sintaxis:

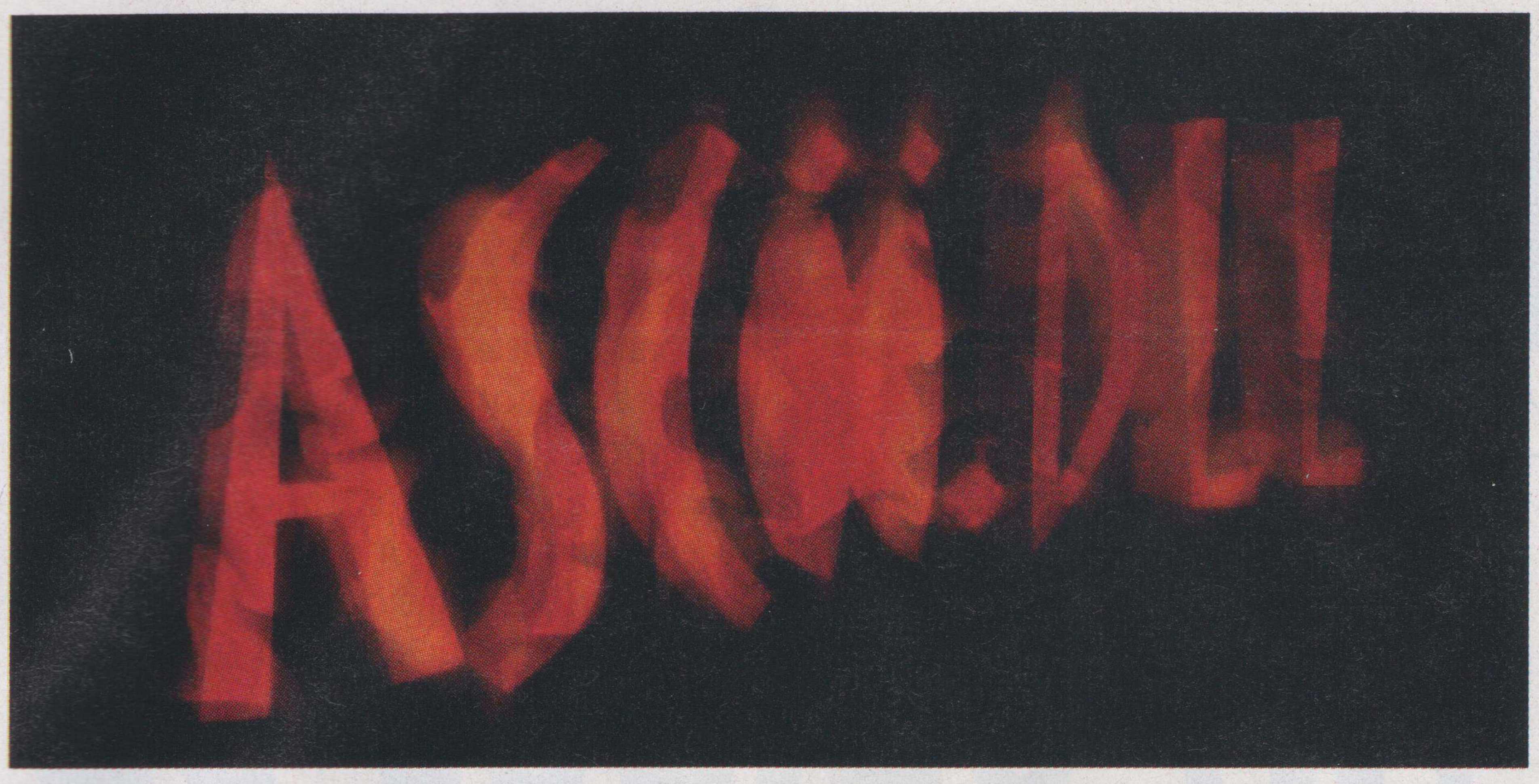
Nombre_tabla[posición] = valor;

Si queremos asignar varios valores a la vez, sólo podremos realizarlo durante la declaración, para lo cual haremos de la siguiente forma:

<tipo de dato> nombre tabla [tamaño de la tabla] = valor1, valor2,...,valor;

El valor del tamaño de la tabla puede ser mayor que el número de valores que hemos indicado posteriormente. Podemos omitir dicho valor, para de esta forma, asignar el tamaño correspondiente al número de valores que aparecen a continuación.

¿Y para qué todo esto? Pues volviendo al caso que nos preocupa, las



En la primera versión de DIV podíamos utilizar las funciones de la librería ASCII.dll para realizar operaciones que se han introducido en DIV 2.

cadenas de caracteres (o STRING), no son más que tablas en las que cada letra de nuestra cadena ocupa un lugar. Por tanto, podremos acceder individualmente a cada una de las letras como si de una tabla se tratara.

La única diferencia respecto a las tablas normales radica en el final de la cadena. Como toda tabla, una cadena tiene una capacidad límite, indicada durante la declaración. Sin embargo, podemos escribir una cadena de caracteres que ocupe menos espacio, con lo que en el espacio sobrante se almacenarán valores no deseados. Para que el compilador de DIV no tome estos valores como válidos, debemos indicar dónde acaba la secuencia de texto. Para ello utilizaremos el valor nulo, es decir, cuando nuestra cadena se acabe, añadiremos al final de ésta un valor de carácter 0.

Todo esto parece complicado, pero es completamente transparente para el programador si utilizamos las funciones que se añaden en la nueva versión de DIV para la manipulación de estas cadenas. Estas funciones, que serán descritas más adelante, manipulan sin ningún tipo de problemas las cadenas, con lo que tan sólo nos debemos preocupar de no manipularlas manualmente para no provocar fallos importantes en nuestros programas.

Veamos pues, para finalizar con esta introducción, cómo podemos definir el tamaño de nuestras cadenas de caracteres y cómo podemos asignarles un valor. Como vimos anteriormente, podemos declarar una cadena simplemente escribiendo *STRING nombre*. En este caso, el número máximo de caracteres que puede almacenar la cadena es de 256. Para variar dicha limitación, bien porque creamos excesivo el tamaño y deseemos ahorrar memoria para nuestro programa, bien porque deseamos utilizar cadenas más largas, debemos utilizar la sintaxis de declaración de tablas:

```
STRING nombre[tamaño];
```

Por tanto, STRING nombre; equivale a STRING nombre[256]; .

Nota: Los miembros de una tabla pueden ser también accedidos a través del paso de una variable que almacene un valor que indique la posición a la que queremos acceder. Esto quiere decir que el código que sigue:

```
BYTE a = 5;
Cadena[a] = "a";
```

Es equivalente a Cadena[5]= "a";

Esto nos permitirá el acceder de forma secuencial a los elementos de una tabla, por ejemplo mediante un bucle cuya variable sea el índice de nuestra tabla.

Dando valores a las cadenas

Para asignar un valor a una cadena, debemos seguir la sintaxis habitual que hemos utilizado para asignar valores a otros tipos de datos. La única diferencia radica en la expresión de la cadena. Ésta debe aparecer entrecomillada (comillas dobles) y si aparece en varias líneas, deben aparecer entrecomilladas cada una de las partes en las que está dividida la palabra y sin ningún tipo de código entre las líneas, excepto comentarios. Veamos un ejemplo:

```
STRING cadena = "DIVmanía";
```

Que es equivalente a:

```
STRING cadena = "DIV"
// cadena de caracteres "manía";
```

Realizando operaciones con cadenas

DIV 2 añade una gran cantidad de operaciones, que nos permite tratar este nuevo tipo de dato como tal, de forma que podamos realizar operaciones aritméticas de todo tipo con él. Todas estas operaciones tienen un equivalente en forma de función, si bien la forma aritmética tiene un inconveniente en lo que se refiere al tamaño de la cadena: no puede sobrepasar nunca los 1024 caracteres en el resultado. Por tanto, debemos prever siempre esta situación en todos los programas que realicemos.

Estas son las operaciones que podemos realizar:

- **Acceso como tabla:** ya descrita anteriormente.
- **Paso como parámetro:** tal vez sea la opción que nos dé más juego. Podemos enviar como parámetro a las funciones que precisen de una cadena de texto, una variable STRING. Con esta nueva capacidad podemos por ejemplo pedir el nombre de un fichero por teclado para posteriormente cargar la imagen asociada a dicho nombre mediante la función load_map. Posteriormente la analizaremos a fondo.
- **Asignación:** además del método visto anteriormente, podemos asignar el valor de dos cadenas entre sí. El código que sigue equivaldría a copiar el contenido de s2 a s1:

  ```
  S1 = s2;
  ```

  La función que equivale a esta operación es *strcpy(s1,s2)*, que recibe como parámetros las cadenas origen y destino. La cadena de origen (s2) puede ser el valor de una cadena, es decir, sería válido: *strcpy(s1,"DIVmanía");*

> Los string son tablas en las que cada letra ocupa un lugar

- **Concatenación:** esta operación equivale a añadir una nueva cadena de texto al final de otra. Es decir, si tenemos dos cadenas s1 y s2 con valor *"DIV"* y *"manía"* respectivamente, la concatenación s1s2 sería "DIVmanía". El orden entre ambos operadores no se puede invertir, ya que s2s1 generaría "maníaDIV". Para realizar esta operación, utilizaremos el operador suma:

  ```
  S1 = string 1 + string 2 + ... + string n;
  ```

  Donde las cadenas pueden ser tanto un valor como una variable. La función equivalente a esta operación es *strcat(s1,s2)*, que nos permite copiar s1 al final de s2. La desventaja de esta operación es que sólo se puede concatenar un elemento al mismo tiempo.
- **Añadir caracteres:** es un caso especial del anterior. Se realiza mediante la sentencia: s1+= "a";

– **Eliminar caracteres:** se pueden suprimir del final de una cadena mediante cualquiera de las siguientes sentencias:

S1-- ; s1-=1 ; s1 = s2-1;

Una variante de esta operación la encontramos en la función *strdel(s1,prin,fin)*; donde s1 es la cadena de la que se quieren eliminar los prin caracteres y los fin caracteres.

– **Comparaciones:** nos permite determinar si una cadena es idéntica a otra o, el orden en el que aparece cada una de ellas alfabéticamente. La sintaxis es la que sigue:
S1 < s2: S1 antes en orden alfabético.
S1 <= s2: S1 antes en orden alfabético o idéntica.
S1 == s2: S1 idéntica a s2.
S1 >= s2: S1 después en orden alfabético o idéntico.
S1 > s2: S1 después en orden alfabético.
S1 <> s2: S1 no idéntico a s2.
Estas expresiones pueden utilizarse en sentencias de tipo IF. Otra opción es la de utilizar la función *strcmp(s1,s2)*; como alternativa. Esta función compara dos cadenas, devolviendo un número positivo si s1 va después de s2 en orden alfabético y negativo si va después. Si son idénticos, el resultado es 0.

Las comparaciones nos permiten determinar las cadenas idénticas

– **Longitud:** mediante la función *strlen(s1)*; podemos determinar la longitud de una cadena. El valor devuelto no es el tamaño de la tabla que lo almacena, sino el de la palabra almacenada.

– **Relleno:** la función *strset(s1,c);* nos permite rellenar la cadena s con el carácter c repetido tantas veces como la longitud de la tabla de la cadena.

– **Posición:** las funciones *strstr(s1,s2)* y *strchr(s1,c)* devuelven la posición donde se encuentran la cadena s2 y el carácter c en la cadena s1 respectivamente.

– **Mayúsculas y minúsculas:** mediante las funciones *lower (s)* y *upper(s)* podemos convertir a minúsculas o mayúsculas respectivamente, todos los caracteres de la cadena s.

– **Cálculo de expresiones:** esta función es un tanto especial, ya que nos permite utilizar DIV como si de una calculadora se tratara. La función *calculate(s)* devuelve el resultado de realizar una operación matemática expresada en la cadena s. Dicha expresión puede ser desde un número hasta una combinación de operaciones básicas de DIV (modula, resta, suma, producto, raíces cuadradas...).
Veamos un ejemplo:

```
A = Calculate ("2+3*4");
// A valdrá 14
  A= Calculate("14");
// A valdra también 14
```

– **Conversión de enteros a cadenas:** mediante la función *itoa(entero)*; podemos convertir cualquier número entero en una cadena de texto. Continuando con el caso anterior, la función *s1 = itoa(A);* asignará a s1 el valor 14.

Encontrando una aplicación

Después de toda esta descripción tan abultada de datos, estamos ya en disposición de crear algo útil con todas estas funciones. Como dijimos al comienzo, podemos tomar datos desde el teclado e introducirlos en cadenas. Pero estos datos se deben leer de alguna forma. Hasta ahora, tan sólo sabemos determinar la pulsación de diversas teclas mediante la función *key()*, lo que nos supondría el realizar una función de selección tan grande para determinar la tecla pulsada, que ocuparía gran parte del tiempo de cálculo en cada frame. Es por ello que DIV posee una variable global predefinida que contiene el código ASCII (código de caracteres internacional) de la última tecla que ha sido pulsada. Esta variable se denomina *ascii*.

Este código ASCII es el que se utiliza para almacenar los diversos caracteres de una cadena y, por tanto, si tomamos dicho valor, tan sólo debemos añadirlo a una cadena como dijimos anteriormente:

```
S1 += ascii;
```

De esta forma podemos realizar una sencilla línea de comandos como la que sigue:

```
PROCESS línea_de_comandos(x,y)
PRIVATE
STRING cadena = ""; // este valor nos indica que se trata de una cadena vacía
  ident;
BEGIN
Ident = write (0,x,y,0,cadena);

LOOP
  IF (key(_backspace))
    Cadena--;
  ELSE
    Cadena += ascii;
  END
  FRAME;
END
// si se sale, borra el texto
  Delete_text(ident);

  END
```

Hemos introducido la detección de la pulsación de la tecla de borrado para que el comportamiento de la línea de comandos sea el que tenemos acostumbrado, y no aparezcan caracteres extraños durante la ejecución. Es por ello que nos debemos siempre detectar el caso especial de la tecla de borrado.

Es importante reseñar que si bien teníamos entendido que un texto no podía variar dinámicamente como lo hacían los contadores numéricos (como vimos en los primeros capítulos con el coche de carreras), en este caso y con el uso de variables, –si disponemos de esta opción–, el código resultante es muy sencillo y se limita al cambio del contenido de las variables asignadas a los identificadores de texto.

Otra posible aplicación de estas funciones es la creación de una sencilla calculadora en línea. Para realizarla, utilizaremos las funciones *itoa* y *calculate*:

```
PROGRAM prueba;

PRIVATE
STRING cadena;
BEGIN
cadena = "";
write (0,0,0,0,cadena);
LOOP
  if (key(_backspace))
    cadena--;
  ELSE
  IF (key(_enter))
    cadena = itoa(calculate(cadena));
  ELSE
    cadena+=ascii;
  END
  END
  FRAME;
END

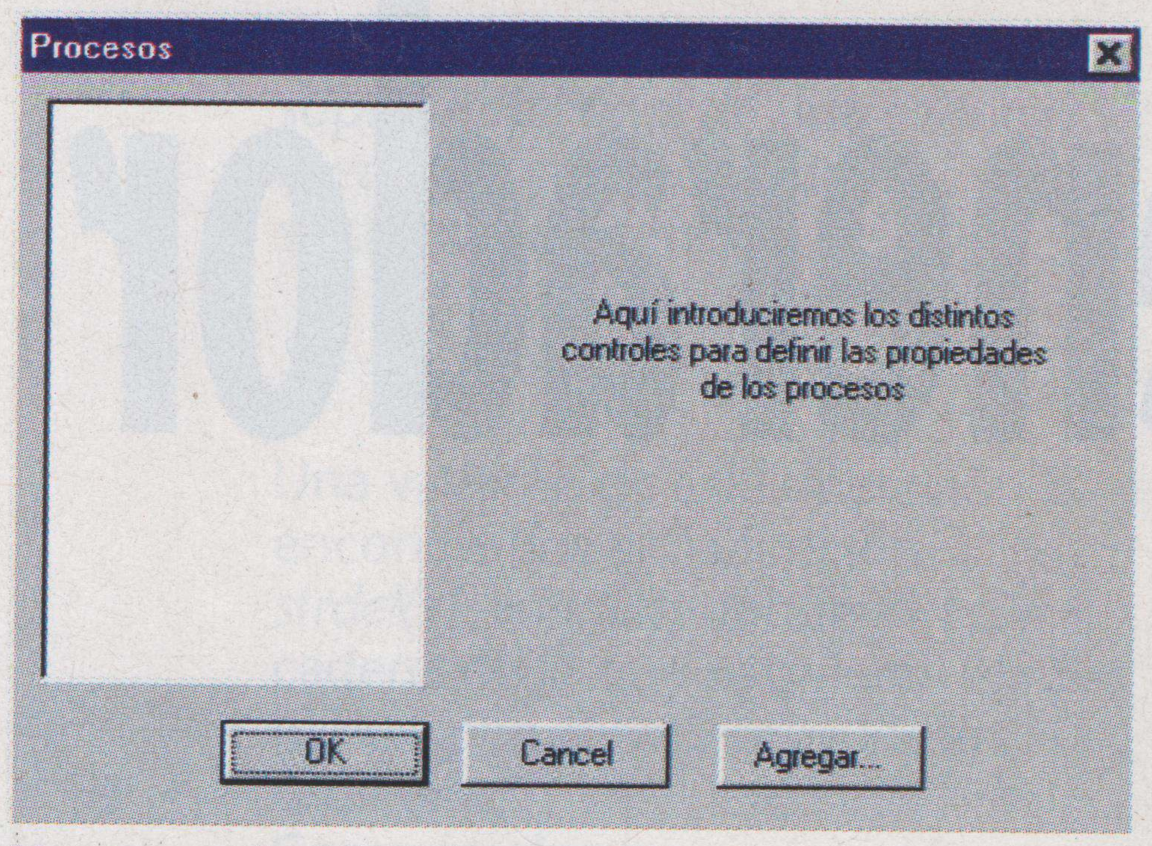
END
```

Conclusiones

Estas son algunas aplicaciones básicas de las cadenas de texto. Todas las opciones anteriormente descritas nos permiten ampliar nuestro rango de acción y realizar nuevas aplicaciones, como las tablas de récord anteriormente descritas, o el famoso juego del ahorcado, o un parser para aventuras gráficas, etc.

Para cualquier tipo de consulta, sugerencia u opinión, podéis dirigiros a la dirección de *e-mail:* *trinidad@arrakis.es.*

Pablo Trinidad



Ventana de procesos.

guardar todos los datos de una ventana en una estructura que se declare globalmente, al igual que en el caso anterior, en el archivo principal de la aplicación o en el del marco principal de la ventana. De esta forma, accedemos sin ningún problema a los datos desde cualquier punto de la aplicación, sin embargo, tenemos una tarea adicional, que es la de actualizar los datos de la ventana de diálogo cada vez que se ejecute. Para ello, utilizaremos un evento o función al que Windows llama cada vez que una ventana se inicia: *WM_INITDIALOG*. En esta función, actualizaremos las variables correspondientes a cada control, para poder mostrar el mismo aspecto que cuando salimos de dicha ventana.

Tenemos que actualizar los datos de la ventana de diálogo cada vez que se ejecute

Veamos pues el procedimiento que hemos seguido para separar dichas opciones, para lo cual utilizaremos la primera variante.

Separando las opciones

Para empezar, debemos declarar una variable en la clase *CMainFrame*, que se corresponderá con el diálogo *CDatosPrincipal*. Para ellos pulsaremos como dijimos el botón derecho sobre el nodo correspondiente a nuestra clase y seleccionaremos la opción *añadir variable miembro*. Declararemos un puntero de la siguiente forma:

```
CDatosPrincipal *datos;
```

Hemos elegido un puntero, para asignar dinámicamente memoria al diálogo, lo que es mucho más correcto que utilizar una variable estática, ya que se le reserva para ella espacio dentro del ejecutable, de forma que desde el inicio hasta el final de la aplicación se está ocupando una memoria que podría estar aprovechándose para otra cosa. Ahora, debemos asignarle memoria en la función *OnCreate*, llamada por Windows cuando se crea la aplicación. Añadimos:

```
Datos = new CDatosPrincipal;
If (Datos == NULL) return –1;
```

De esta forma nos aseguramos de que la operación se ha realizado con éxito.

Este procedimiento es el que seguiremos para cualquier diálogo que creemos en un futuro. Debemos recordar que este código no funcionará a menos que se le añada la carga del fichero de cabecera donde está declarada la clase *CDatosPrincipal* en el fichero del marco de aplicación:

```
#include "datosprincipal.h"
```

Debemos ahora separar la opción de menú en dos. Para ello, conservaremos el identificador de evento que tenía asociada la anterior opción de menú y lo reutilizaremos en la nueva opción que crearemos, donde se seguirá llamando al diálogo de datos principales. De esta forma, nos ahorramos gran cantidad de variaciones. Para ello utilizaremos el editor de recursos como especificamos en el número anterior y obtendremos dos submenús distintos:

- **Opciones:** donde están ubicadas todas las posibilidades del sistema, como los datos principales, la lista de procesos del juego...
- **Crear archivo:** donde se generará el archivo con el código definitivo de nuestro juego.

Dentro del *ClassWizard*, definiremos la función asociada a cada una de las opciones del menú. En la función asociada a la llamada del diálogo, debemos cambiar el contenido anterior (llamada estática) por una llamada tan simple como:

```
datos->DoModal();
```

Para el generador de códigos, debemos hacer algunos cambios con respecto al anterior sistema. En el número anterior, implementamos la grabación dentro de la función *OnOk()* del diálogo de datos, que se ejecutaba si se salía de la ventana de diálogo pulsando "Ok". Si copiamos el código anterior en la función asociada al evento de creación de archivo, tan sólo debemos realizar algunos pequeños cambios. Estos cambios se corresponden con la localización de las variables de datos. Estas variables ya no son locales a la clase, sino que se encuentran accesibles a través del puntero *datos*. Por tanto, el acceso debe realizarse como sigue:

```
Datos->variable;
```

Tan sólo nos falta un pequeño detalle. Debemos asegurarnos que los datos se conservan entre llamada y llamada. Para ello debemos guardarlos justo antes de salir. Esto lo haremos en la propia función OnOk(), mediante una llamada a la ya conocida función UpdateData(TRUE), que se encarga de almacenar en toda y cada una de las variables miembros, el dato almacenado en sus controles correspondientes.

Ya disponemos pues de un sistema separado.

Introduciendo el nombre del programa generado

Para poder especificar el nombre del programa que generaremos y no estar restringidos al propio uso del archivo *result.prg*, podemos realizar una operación semejante a la realizada para la selección del fichero de imágenes en el diálogo de datos básicos. Abriremos una de las ya famosas ventanas de guardar fichero, y tomaremos el nombre del fichero resultante y lo crearemos para posteriormente operar con él:

```
void
CMainFrame::OnCrearprograma()
  {
    FILE *salida;
    CFileDialog
    file(false,"prg",NULL,OFN_HIDEREA-
    DONLY|OFN_OVERWRITEPROMPT,
      "Programa
    (*.prg)|*.prg||",NULL);
    if (file.DoModal()==IDOK)
    {
    if ( (salida =
    fopen(file.GetFileName(),"w")) ==
    NULL)
      {
        MessageBox("Error en la
    apertura del fichero de
    salida","Error");
      else
      {....
```

De esta forma sólo se grabará si pulsamos OK en la ventana de diálogo, y habremos conseguido nuestro primer objetivo.

Añadiendo nuevas capacidades

Una vez separados estos dos elementos, ya podemos plantearnos la posibilidad de ampliar nuestro entorno, añadiéndole una capaci-

dad que sin duda es bastante interesante: la creación de procesos.

Para realizar esta tarea, debemos prever el diseño que realizaremos de la misma para poder atacar frontalmente su creación. En un primer término, tendremos una ventana de diálogo a la cual se accederá desde el menú de opciones. En dicha ventana aparecerá una caja de lista, donde como su propio nombre indica, se listarán todos los procesos que queremos crear. Para crear un nuevo proceso, utilizaremos un botón donde podamos agregarlos, de forma que si se pulsa, aparecerá una nueva ventana de diálogo donde podremos establecer el nombre del proceso. Utilizaremos el segundo procedimiento de los listados al comienzo, es decir, utilizaremos una clase externa donde almacenaremos los datos de cada proceso.

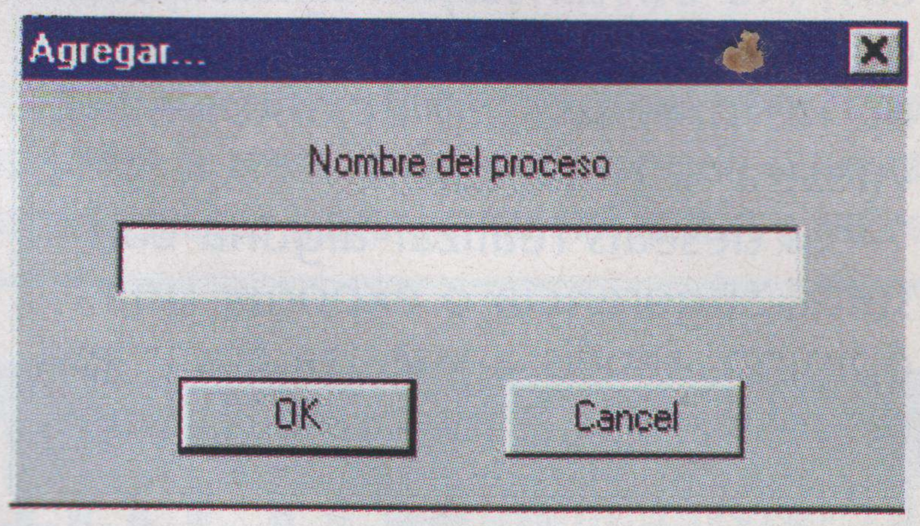
Dentro del propio diálogo, podemos especificar todas las propiedades que deseemos para cada uno de los procesos. Para ello, dispondremos de una serie de opciones que aparecerán a la derecha de dicha lista y que variarán según el proceso elegido en la propia lista. Si bien no lo realizaremos en este número, indicaremos la forma de realizarlo, ya que es muy sencillo con la estructura que vamos a definir.

Procedimiento para la creación del diálogo de procesos

Creamos una nueva ventana de diálogo, dentro de la cual introduciremos una caja de lista y un nuevo botón cuyo título será *agregar*. Mediante el *ClassWizard* crearemos la clase correspondiente a este nuevo diálogo que se llamará *CProcesosDlg*. Definimos ahora pues varios eventos que nos ayudarán a crear el comportamiento deseado tal y como podéis ver en la tabla 1.

Y debemos establecer las variables que almacenarán los valores de cada control como figura en la tabla 2.

Para guardar las propiedades de cada proceso, debemos crear



Ventana para agregar el nombre de un nuevo proceso.

Tabla 1

Control	Evento	Acción
Botón agregar	BN_CLICKED	Abre el diálogo de agregar cuando se pulsa el botón
Lista	LBN_SELCHANGE	Cambia las propiedades que aparecen en la ventana cuando se selecciona otro proceso
Botón OK	BN_CLICKED	Llama a la función UpdateData() para grabar todos los datos en sus variables correspondientes
Ventana	WM_INITDIALOG	Establece los valores de cada uno de los procesos en la ventana

Tabla 2

Control	Tipo	Variable
IDC_LISTA	CString	m_Proceso
IDC_LISTA	CListBox	m_ListaProcesos

una estructura o clase, lo que haremos rápidamente mediante la opción *Nueva clase* del menú "Insertar". Dentro de las opciones, elegiremos como tipo clase genérica y le pondremos el nombre *CProcesosDatos*. Esta clase es muy sencilla y tan sólo, contendrá una variable pública de tipo *CString* que llamaremos *m_Nombre*.

Para poder utilizar esta estructura debemos crear una pila de esta estructura, es decir, una estructura dinámica que nos permita crearlas y destruirlas a nuestro antojo. Esto es bastante más complejo y, una solución bastante rápida y funcional es crear un *array* o matriz de estas estructuras con un tamaño fijo. Dicho tamaño lo fijaremos en el momento de compilación, mediante una variable constante global o una sentencia como la que sigue:

```
#define MAX_PROCESS 10
```

Y dentro de la clase *CProcesosDlg* escribiremos:

```
class CProcesosDlg{
Public:
...
CProcesosDatos datos
[MAX_PROCESS];
Int contador;
...
};
```

La variable contador la utilizaremos para controlar en todo momento el número de estructuras de datos que se encuentran almacenadas, y evitar además añadir más procesos de los permitidos por la estructura. Es por ello que debemos iniciarla a 0 en el constructor de la clase.

Creando el diálogo de agregar

Tan sólo nos resta diseñar nuestro diálogo, donde agregaremos el nombre del proceso que deseamos crear. Para ello, utilizaremos de nuevo el editor de recursos, donde agregaremos un nuevo diálogo.

Como en todos los diálogos creados hasta el momento, en primer lugar introducimos los controles, que en nuestro caso, consistirá en una caja de edición donde introduciremos el nombre del nuevo proceso y un texto estático que nos indique el título.

> Para diseñar nuestro diálogo utilizaremos el editor de recursos

En un segundo paso, creamos la clase asociada a la ventana de diálogo, a la que llamaremos *CAgregar* y le asociamos al valor del texto de la caja de edición una variable de tipo *CString* a la que llamaremos *m_Nombre*.

Por último escribiremos el código necesario. Sólo modificaremos el evento ONOK, correspondiente a la pulsación del botón de OK de nuestra ventana. De esta forma, conseguiremos que se conserve el valor introducido en la caja de edición en la variable *m_Nombre*, que posteriormente podremos leer para introducirlo en el cuadro de lista de la ventana de procesos.

Pero debemos tener en cuenta que no todos los nombre de procesos que introduzcamos son válidos. Es importante que no tengan espacios en el nombre. Los espacios pueden sustituirse por el carácter de subrayado. Por eso, el código resultante en la función *OnOK()* será:

```
void CAgregar::OnOK()
{
    UpdateData(TRUE);
    char *temp;
    temp = new
char[m_Nombre.GetLength()];
    int i = 0;
    while
(i<m_Nombre.GetLength())
    {
        if (m_Nombre[i]==' ')
            temp[i]='_';
        else
            temp[i]=m_Nombre[i];
        i++;
    };
    temp[i] = 0;
    m_Nombre = temp;
    UpdateData(FALSE);
    CDialog::OnOK();
}
```

Es importante realizar la última llamada a *UpdateData()*, ya que si no la realizamos, no se conservan los valores en la clase de forma correcta.

Ahora podremos crear nuevas opciones e integrarlas en el escritorio

Escribiendo el código

Ya sólo nos queda escribir el código del comportamiento de la clase *CProcesosDlg*:

```
void CProcesosDlg::OnOK()
{
    // Primero debemos guardar
los datos en las variables
    UpdateData(TRUE);
    CDialog::OnOK();
}

void CProcesosDlg::
OnCambioProceso()
{
```





Nuevo aspecto de la barra de menú.

```
    // aquí introduciremos los cambios en los controles de las propiedades
    // asociadas al proceso seleccionado
}

void CProcesosDlg::OnAgregar()
{
    CAgregar agregar;
    if (agregar.DoModal()==
IDOK)
    {
        if (contador < MAX_PRO-
CESS)
        {
            m_ListaProcesos.AddString
(agregar.m_Nombre);
            datos[contador].
m_Nombre = agregar.m_Nombre;
            contador++;
        }
    }
}

BOOL CProcesosDlg::
OnInitDialog()
{
    CDialog::OnInitDialog();

    for (int i = 0; i < contador;
i++)
    {
        m_ListaProcesos.AddString
(datos[i].m_Nombre);
    }

    return TRUE;
}
```

Como podemos apreciar, este código es bastante simple y tan sólo tiene una función no utilizada hasta el momento: *AddString()*. Mediante la misma, podemos añadir una cadena de texto al cuadro de lista, devolviéndonos un índice que podremos utilizar más adelante para asociarle datos mediante una función denominada *SetItemData()*. Esta asociación de datos nos permite vincular una estructura de cualquier tipo a cada una de las cadenas que aparecen en el cuadro de lista. Gracias a esto podemos asociar a cada elemento una estructura del tipo *CProcesosDatos* que contenga los valores de las propiedades de cada proceso, y se graben en la estructura global si se pulsa el botón OK y no se grabe si se pulsa Cancelar. No obstante, y por cuestiones de simplificación, esta opción la dejamos a la elección del lector. Hemos optado por agregar directamente el dato en la estructura global e incrementar el contador en consecuencia.

Modificando la creación de ficheros de salida

Para finalizar, debemos permitir la salida por fichero además de la nueva opción de creación de procesos. Para ello, debemos alterar la función correspondiente al evento de la opción de menú de *Crear archivo* y añadirle por ejemplo el siguiente código:

```
int max = Procesos->contador;
for (int i = 0;i < max; i++)
{
    fprintf(salida,"PROCESS %s()\
n",Procesos->datos[i].m_Nombre);
    fprintf(salida,"BEGIN\nLOOP\
n\tFRAME;\nEND\nEND\n\n");
}

// final del fichero
fprintf (salida,"END");
```

Con esto concluimos el desarrollo.

Conclusión

Hemos desarrollado una nueva estructura en nuestro programa que nos permite crear nuevas opciones e integrarlas en el entorno sin ningún problema y con extrema facilidad. No hemos integrado las propiedades de los procesos dentro del diálogo, sin embargo, hemos establecido una estructura que nos permite hacerlo sin ningún obstáculo. Es evidente que el sistema utilizado puede ser mejorado, sobre todo en la limitación que supone tener un número máximo de procesos fijo, mediante el uso de estructuras de almacenamiento dinámicas. No obstante y al igual que lo que ocurre con las propiedades de los procesos, lo dejamos a vuestra libre elección.

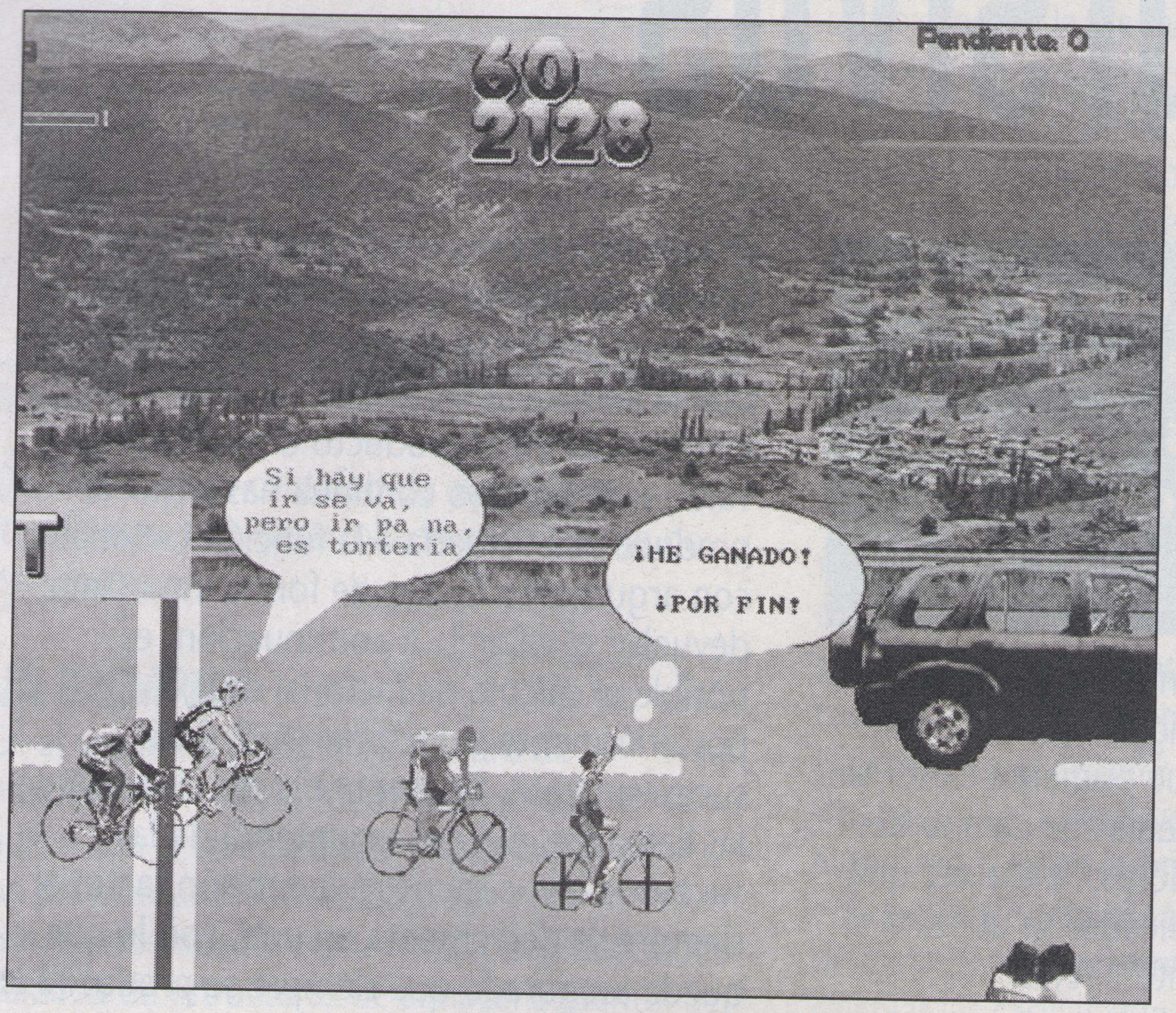
Si deseáis realizar alguna consulta, sugerencia o exponer una opinión, tenéis a vuestra completa disposición la siguiente dirección: trinidad@arrakis.es

Pablo Trinidad

DIV developer

NÚMERO 9

Sumario

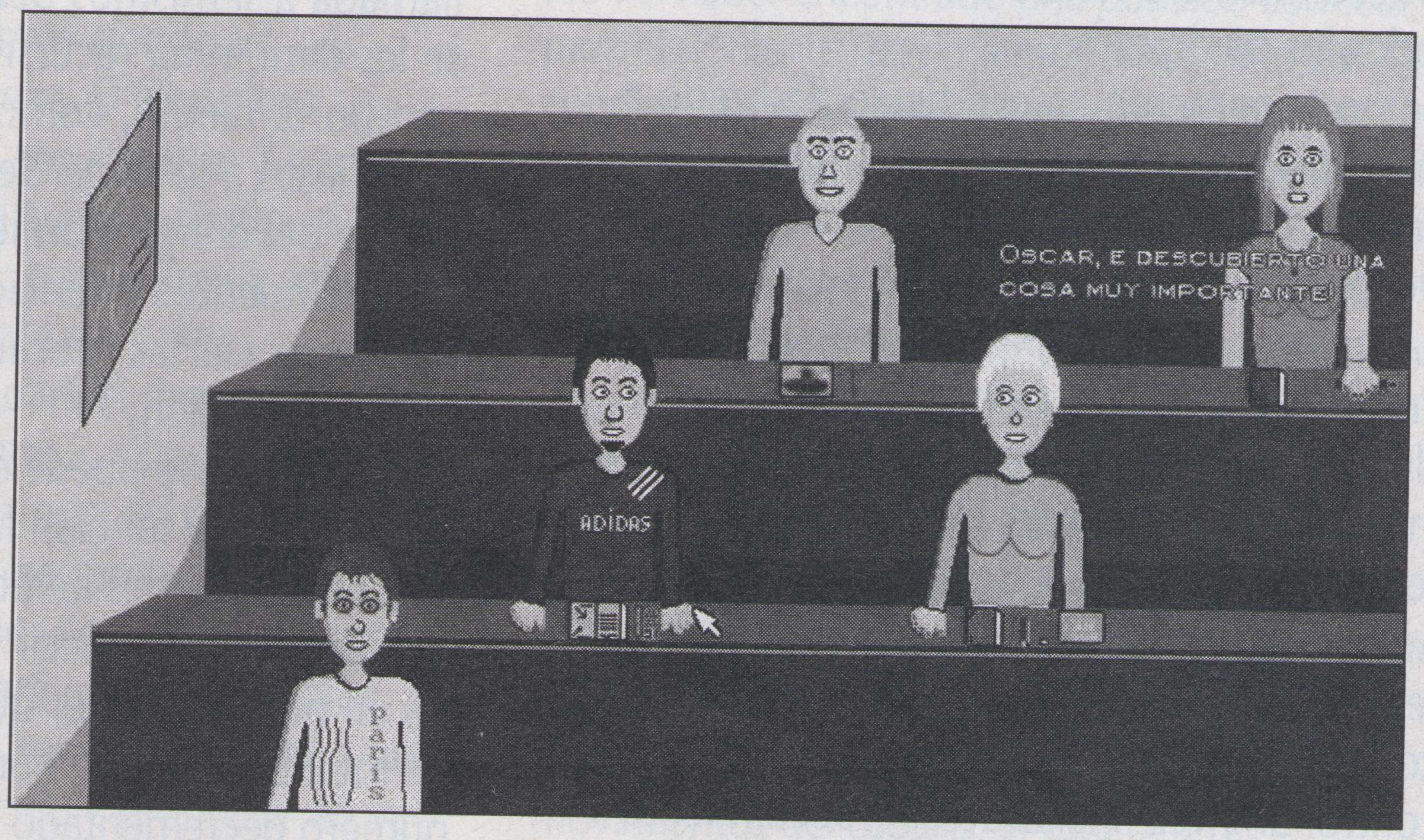


Curso de programación y concurso

Continuamos con nuestros cursos de programación habituales siguiendo los capítulos ya iniciados. Como siempre, esperamos que os sean de utilidad.
En cuanto a nuestro concurso, por supuesto sigue en pie con los mismos premios. Ya sabéis: 25.000 pesetas para el ganador y 20.000 para los otros dos juegos elegidos. Esperamos con impaciencia recibir vuestros juegos.
Aquí tenéis los ganadores de este número. Esperamos que os gusten. En este suplemento encontraréis un breve comentario del juego y un extracto de su código fuente que, no os olvidéis podéis encontrar entero en el CD-Rom si abrís los ficheros pertinentes. Seguro que la visión y el estudio de estos códigos os ayudará a mejorar algunos aspectos de vuestros propios trabajos de programación.

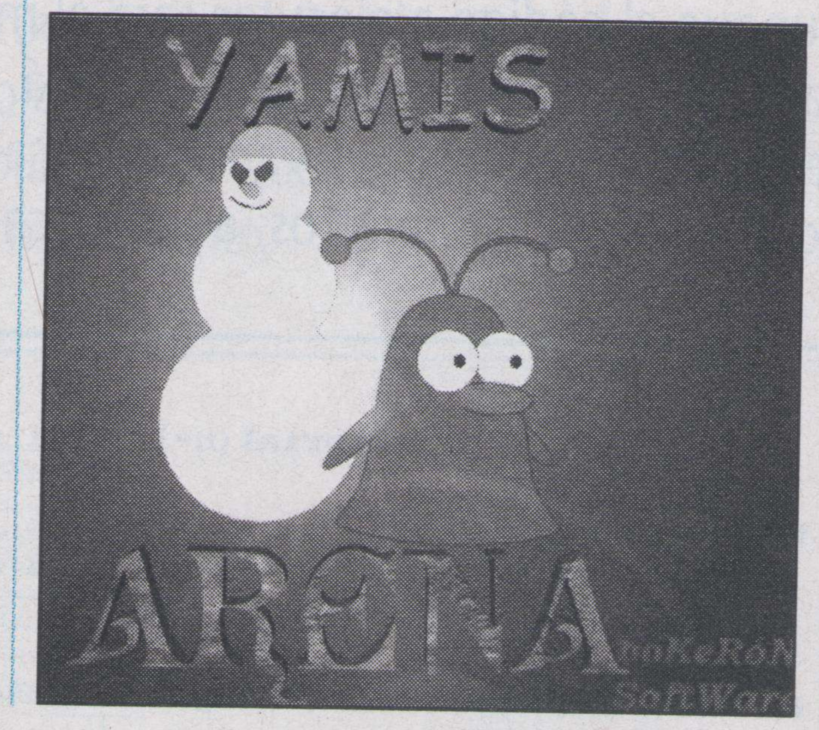
Vuestras sugerencias

Queremos haceros saber que todas las ideas, críticas o sugerencias para mejorar esta revista serán bienvenidas por parte de esta redacción. Si tenéis inquietud por algún tema que no tocamos demasiado, si consideráis que alguna sección de esta revista debe desaparecer o debe ser cambiada por otra, incluso si os animáis a proponernos artículos realizados por vosotros/as, que creéis pueden ser aprovechados por otros usuarios/as, no dudéis en poneros en contacto con nosotros. Todas las sugerencias serán atendidas y tenidas en cuenta. Os necesitamos para mejorar.
Para mandarnos los juegos o cualquier sugerencia sobre la publicación, podéis hacerlo por e-mail, correo normal o incluso entregarlos a mano en la dirección de la redacción:



Nuestra dirección:
divmania@prensatecnica.com

Divmanía
C/ Alfonso Gómez, 42
Nave 1-1-2
28037, Madrid, España



Para los juegos del concurso

No os olvidéis meter vuestros datos completos cuando mandéis juegos al concurso. Lo decimos porque hemos recibido algunos en los que falta la ciudad, el teléfono o directamente la dirección entera. Para hacer más cómoda la tarea, en el CD que acompaña la revista hay una ficha que debéis incluir como fichero de vuestro juego para evitar que haya problemas. Esta ficha se incluirá en la revista cuando hagamos otro especial de juegos DIV con todos vuestros datos.

También recordaros que es vital que introduzcáis un fichero de texto que contenga la presentación de la historia, vuestras fuentes de inspiración, las características del juego, su instalación y, sobre todo, el código fuente.

Destacamos

En nuestro CD incluimos los juegos que han resultado ganadores en este número y, cómo no, sus respectivos códigos para que veáis todo el proceso de creación. Además, unas cuantas demos y programas que nos han enviado los lectores. Si tenéis algo que pueda resultar de utilidad, enviárnoslo y lo meteremos en el próximo CD.

Algoritmos recursivos

La recursividad es una técnica de programación que permite obtener gran cantidad de algoritmos de forma sencilla, y aunque la eficacia de éstos es, en muchos casos, baja, esta técnica sigue siendo útil como punto de partida para el diseño de algoritmos iterativos.

Cuando se hace referencia a un procedimiento o función sus parámetros se transfieren a través de la pila. Previamente a que se produzca la ejecución del código de un procedimiento o función, los parámetros se agrupan en la pila en el orden en que están declarados. Antes de devolver el control, el procedimiento o función elimina de la pila todos los parámetros.
Como se comentó en el artículo dedicado a procedimientos y funciones, los parámetros pueden transferirse por referencia o por valor. De forma simplificada, cuando se utilizan parámetros por referencia se apila un puntero que contiene la dirección actual de almacenamiento del parámetro, lo que permite modificar su contenido. Cuando un parámetro se transfiere por valor, se apila el valor actual. Además, se almacena en la pila la dirección de retorno, que no es más que el punto del código del programa al que se transferirá el control una vez finalizado el procedimiento que se invocó.

RECURSIVIDAD

El proceso que se describió con anterioridad no es incompatible con la situación en la que un procedimiento o función tiene una llamada a sí mismo dentro de su declaración. Aunque el código a ejecutar será el mismo, cada vez que se ejecute, éste lo hará teniendo en cuenta sus variables locales actuales y sus parámetros, aunque cada vez que se invoca al procedimiento se crean las anteriores variables con igual identificador, los posibles conflictos que pudieran existir se resuelven mediante las reglas de campo de validez de los identificadores: "cada vez que se referencia a un identificador se utilizará el creado más recientemente". A los procedimientos y funciones que tienen una llamada a sí mismos dentro de su declaración se les denomina recursivos. Su comportamiento, como se pondrá de manifiesto a lo largo de este artículo, no difiere del comportamiento habitual de los procedimientos y funciones, aunque habrá que tener ciertas precauciones a la hora de su implementación.
En la figura 1 se puede observar una definición recursiva de la función factorial junto con sus implementaciones iterativa y recursiva. La definición viene a decir lo siguiente: el factorial de un número mayor que cero es el resultado de multiplicar dicho número por el factorial del número decrementado en 1, mientras que el factorial del número 0 es 1. La recursividad de la definición es evidente, pues para definir la función factorial se recurre a la misma función. Es importante señalar que el caso del número 0 hace que dicho algoritmo sea capaz de terminar, pues mediante las definiciones recursivas es posible construir algoritmos que no finalicen, por tanto, siempre que se implemente uno de estos algoritmos será necesario buscar una condición que a lo largo de las llamadas recursivas se verifique y que sirva para detener las llamadas al procedimiento o función.
En la figura 2 puede observarse la sucesión de llamadas que se producen a la hora de llamar a la función factorial con el parámetro 4. Como era previsible, la operación producto que aparece en el cuerpo de la función queda suspendida momentáneamente, hasta que se resuelve la llamada a la función factorial con el argumento decrementado en 1, a su vez, cuando se intenta ejecutar dicha función de nuevo el producto quedará suspendido hasta que la llamada a la función factorial involucrada en el producto devuelva un valor; este proceso continúa hasta que se produce una llamada a la función factorial con argumento 0 que, de forma inmediata, devuelve el valor 1. A continuación, es posible el cálculo de factorial de 1, posteriormente el factorial de 2 y así sucesivamente. Se asegura la terminación de las llamadas, pues el argumento de las sucesivas invocaciones a la función factorial siempre se decrementa en uno, con lo que queda asegurado que siempre se producirá la llamada a *factorial(0)*.
La versión iterativa para la función factorial no hace más que almacenar en la variable *acum*, los resultados intermedios que se producían en la versión recursiva. Como se mostrará en el artículo, los algoritmos recursivos se pueden transformar en algoritmos iterativos.
En la figura 3 se muestra la definición y los algoritmos iterativo y recursivo para la función de Fibonacci; en este ejemplo se pone de manifiesto que la versión iterativa es, con mucho, más eficiente que la versión recursiva, pues en ésta cada llamada a la función provoca dos nuevas llamadas y así sucesivamente, por lo que el número de invocaciones crece exponencialmente. Este ejemplo no debe hacer pensar que la recursión es una mera curiosidad, pues es posible comenzar el diseño de un algoritmo utilizando una solución recursiva. Si tras su estudio se comprueba que no es factible su uso debido a las restricciones de tiempo o espacio impuestas a éste, es posible implementar una pila para recoger los resultados intermedios y proceder a implementar el algoritmo de forma iterativa.

$$factorial(n) = \begin{cases} n*factorial(n-1), & si\ n > 0 \\ 1, & si\ n = 0 \end{cases}$$

```
function factorial (n : integer) : integer ;
 begin
  if n= 0
   then
    factorial := 1
   else
```

FIGURA 1.

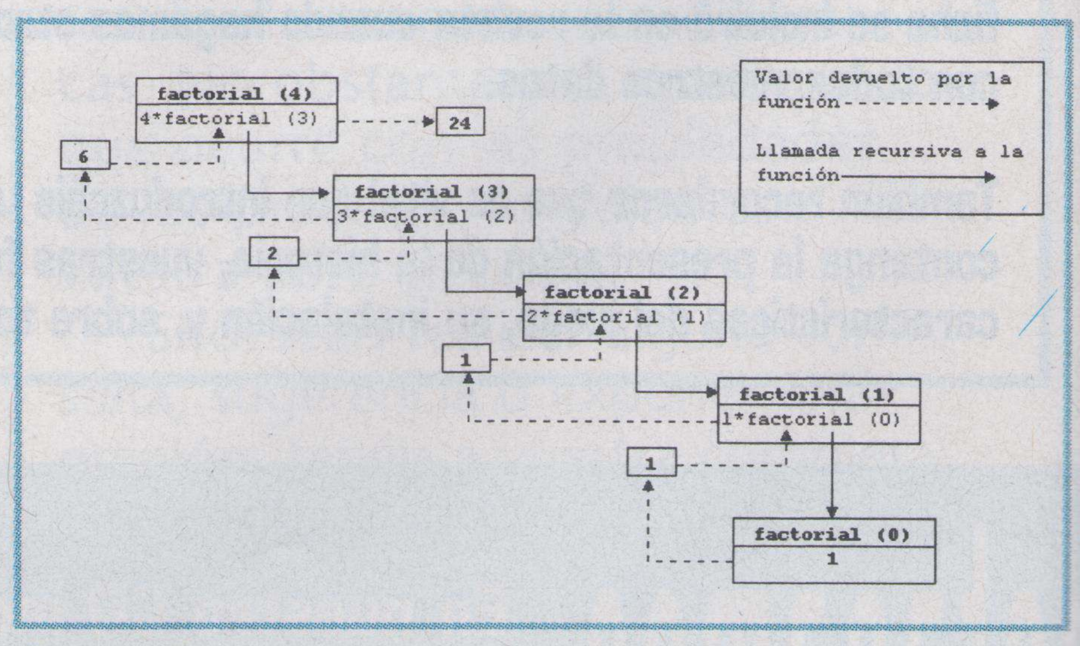


FIGURA 2.

```
function Fibonacci (n : integer) : integer ;
 begin
  if n = 0
    then
     fibonacci := 0
    else
     if n= 1
       then
        fibonacci := 1
       else
        fibonacci := fibonacci(n-1) + fibonacci (n-2) ;
 end ;

function Fibonacci (n : integer) : integer ;
 var
  resul, anterior, i : integer ;
 begin
  resul := 1 ;
  anterior := 0 ;
  i := 1 ;
  while i < n
       do
        begin
          resultado := resultado + anterior ;
          anterior := resultado - anterior ;
          i := i + 1 ;
        end ;
  fibonacci := resultado ;
 end ;
```

```
Definición de la función de
        Fibonacci

fibonacci(0)= 0

fibonacci(1)= 1

fibonacci(n+1)= fibonacci(n) +
                fibonacci(n-1)
```

FIGURA 3.

De hecho, gran cantidad de algoritmos iterativos se han desarrollado inicialmente de esta forma.

UN EJEMPLO

En el siguiente ejemplo se pondrá de manifiesto la utilidad de la recursividad aplicada a la búsqueda de una solución para un problema que no se basa en procedimientos de cálculo ni en una selección cuidada de estructuras de almacenamiento. Se trata de recorrer un tablero de *n* casillas de ancho por *n* de largo utilizando los movimientos de un caballo de ajedrez situado inicialmente en una casilla dada.

En principio, no es posible encontrar una fórmula de cálculo que haga corresponder a cada casilla un número que represente el orden en que tal casilla ha sido seleccionada durante el camino recorrido por el caballo. A falta de dicha fórmula se va a intentar situar en una de las casillas a la que es posible moverse, realizando este proceso repetidas veces. Es evidente que la mayoría de las veces no será posible continuar en un determinado punto, quedando casillas del tablero por las que el caballo aún no ha pasado.

Una solución consiste en guardar de alguna forma las casillas que el caballo podría haber utilizado para sus anteriores movimientos y no utilizó, intentando seleccionar una de dichas casillas, realizando nuevos movimientos hacia delante y así poder encontrar la solución. Este tipo de algoritmo se denomina "ensayo y error" y como se podrá comprobar la recursividad será una ayuda muy importante.

Se comenzará la solución del problema seleccionando las estructuras de datos a utilizar. Como es obvio, el tablero estará representado por un array bidimensional de *n* filas por *n* columnas cuyo tipo base será un entero positivo. El valor 0 de una componente representará que la casilla correspondiente no ha sido utilizada; cada vez que una casilla sea visitada por el caballo se le dará el valor del número de movimiento correspondiente (hay n^2-1 movimientos a realizar en total), de tal forma que al finalizar el algoritmo puedan seguirse el orden de los movimientos.

La siguiente cuestión a resolver consiste en establecer la representación para los movimientos del caballo y establecer un orden sobre ellos. En la figura 4 se muestran los posibles movimientos del caballo numerados según el sentido de las agujas del reloj. Para calcular los posibles movimientos del caballo se utilizará un array de 8 elementos de tipo registro con dos componentes enteros que indican los incrementos y decrementos a aplicar sobre la posición actual del caballo. Hay que indicar que estos movimientos son todos los posibles, no los factibles, pues si el caballo está situado suficientemente cerca de los límites del tablero no será posible realizar alguno de los movimientos dentro de dichos límites; de igual forma no serán posibles los movimientos a las casillas que se ocuparon previamente.

El proceso que seguirá el algoritmo que se está diseñando será de la siguiente forma: el algoritmo conoce la primera casilla donde se encuentra situado el caballo y a través del array de incrementos y decrementos es capaz de calcular las casillas que le es posible ocupar en el siguiente movimiento, además, éstas siguen el orden del índice del array donde se almacenan los incrementos. El algoritmo intentará probar para el movimiento 2 con la primera de las casillas calculadas según se describió anteriormente; tras ocuparla (dando el valor 2 a la componente del array correspondiente) continuará el proceso de ocupación de las siguientes casillas. Este proceso continuará hasta que se hayan ocupado la totalidad de las casillas (el número de movimientos realizados será de n^2-1) con lo que el algoritmo terminará con éxito, o bien, al intentar ocupar la casilla correspondiente a un movimiento *k* no le es posible progresar en su recorrido (ninguno de los movimientos permitido es factible), el algoritmo tendrá que deshacer la ocupación de la casilla correspondiente al movimiento *k-1* y volver a intentar ocupar una casilla para el movimiento *k*. Este proceso puede involucrar a la casilla ocupada en el movimiento *k-2*, si ninguno de los movimientos factibles realizados en el movimiento *k-1* tienen como resultado el poder ocupar una casilla en el movimiento *k*; este proceso puede provocar un efecto dominó hacia movimientos anteriores. El algoritmo terminará cuando se han analizado todas las posibles combinaciones (no se ha encontrado ninguna solución) o bien cuando se encuentre la primera solución (se cubrieron todas las casillas del tablero).

Para implementar el algoritmo se ha decidido que el array que representa el tablero y el array de incrementos sean variables globales, pues es inadmisible, dado su tamaño, que cada vez que se llame al procedimiento recursivo éstos pasen como parámetros. Sí que formarán parte de los parámetros del procedimiento el número de movimiento que se va a realizar y la posición desde la que se debe realizar dicho movimiento pues, a partir de ésta, se calculan los posibles movimientos hacia la siguiente casilla. Falta decidir cómo es posible comunicar al procedimiento que provoca la llamada recursiva si se ha realizado con éxito el movimiento que se solicita a través de ésta. Bastará con pasar por referencia un parámetro de tipo booleano.

La variable *i_incremento* será utilizada para recorrer el array que contiene los incrementos, para calcular los posibles movimientos desde la coordenada que le es dada a modo de parámetro. La variable *pos_prueba* contiene la coordenada desde la que realizará el siguiente movimiento, y la variable *aux_ok* permitirá conocer el resultado del siguiente movimiento, correspondiente a la llamada recursiva. El cuerpo del procedimiento no es más que un bucle que tiene como condición de salida el haber intentado los ocho posibles movimientos desde la posición que le es pasada como parámetro o bien el haber encontrado una posición correcta para el siguiente movimiento (ambas son las condiciones de parada del procedimiento recursivo), o bien el haber realizado el último movimiento.

En el cuerpo correspondiente a la instrucción iterativa se puede observar cómo se procede a ocupar la primera casilla que se encuentra disponible, anotando dicho movimiento en el tablero, y realizando una llamada recursiva para que se realice el siguiente movimiento a partir del movimiento realizado. Esta llamada recursiva puede devolver el valor *false* en el parámetro *ok_aux* indicando que no pueden progresar los movimientos del caballo. Si esto ocurre, entonces el procedimiento intenta realizar un nuevo movimiento (esto es posible puesto que en la variable *i_incremento* se almacena la posición del último incremento utilizado para el cálculo del movimiento) y seguir realizando la llamada recursiva. Pero puede ocurrir que no queden más movimientos por realizar; en este caso, el procedimiento devolverá en el parámetro *ok* el valor *false*, de tal forma que desde el punto en el que fue llamado se intente un nuevo movimiento.

Posición	Incrementos
1	(2, -1)
2	(2, 1)
3	(1, 2)
4	(-1, 2)
5	(-2, 1)
6	(-2, -1)
7	(-1, -2)
8	(1, -2)

Valores para el array de movimientos

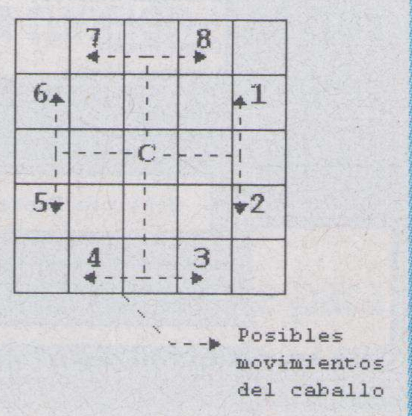


FIGURA 4.

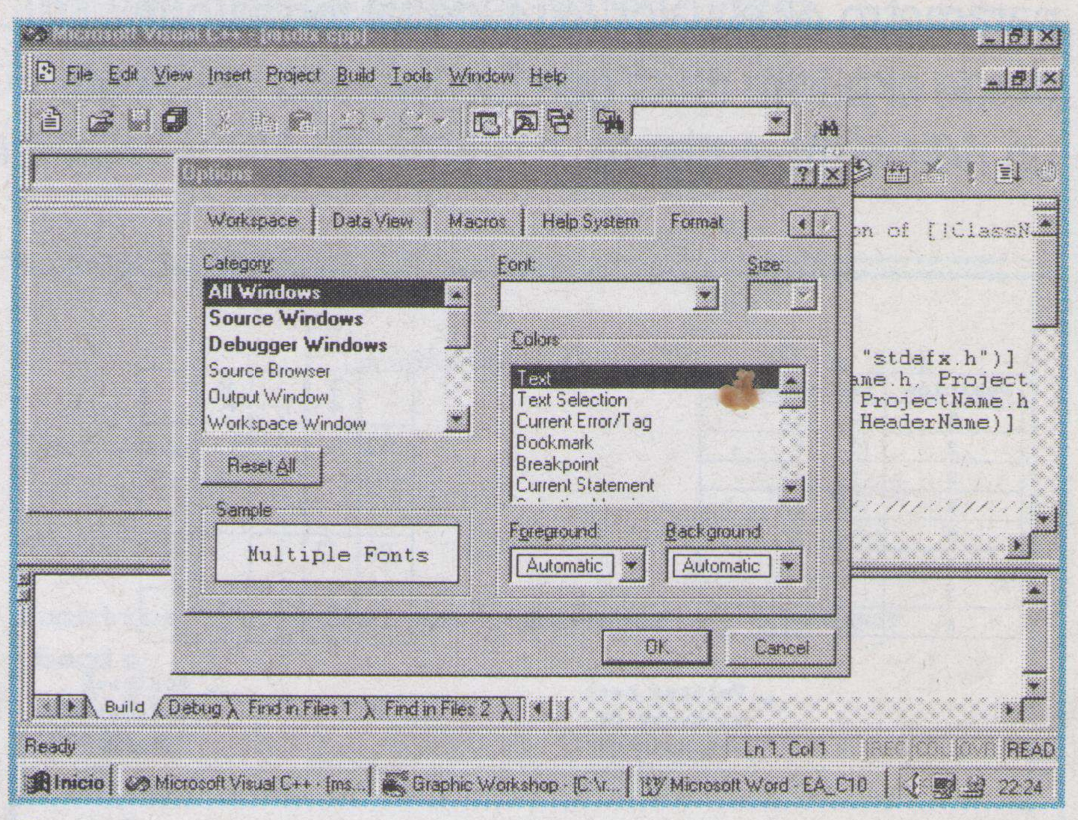
Cómo usar punteros en C

En este artículo se presenta el primer capítulo de un pequeño cursillo de C que se va a dedicar al tema de los punteros, un tema muy interesante e importante que sirve para conocer y poder realizar aplicaciones potentes y rápidas.

La mayoría de la gente considera los punteros como uno de los temas más difíciles del lenguaje C. Existen diferentes razones para que esto ocurra. Primero, el concepto de puntero lleva emparejado otro, el de dirección, que puede ser nuevo para los programadores, ya que no es de uso común en lenguajes como BASIC o Pascal. Segundo, los símbolos utilizados en la notación de punteros no son tan claros como debieran; por ejemplo, se utiliza el mismo símbolo para dos propósitos diferentes aunque relacionados. Conceptualmente, sin embargo, los punteros no son realmente tan complicados y con un poco de práctica los símbolos comenzarán a tener sentido. En otras palabras, los punteros pueden ser difíciles, pero no demasiado. Nuestro objetivo en este minicurso dedicado a los punteros es desmitificarlos, explicando tan claramente como sea posible qué hacen y cómo trabajan.

Para cumplir este cometido, empezaremos lentamente, en un intento de asegurar la comprensión de los conceptos básicos antes de profundizar en el estudio de los punteros en situaciones más avanzadas.

Antes de ver ejemplos de programas que utilicen punteros, vamos a realizar un examen global de este concepto y de los motivos de su empleo.



En un compilador actual de Windows, todo es configurable.

¿QUÉ ES UN PUNTERO?

Un puntero proporciona una vía de acceso a una variable (o a tipos de datos más complejos, como por ejemplo un *array*) sin referirse directamente a la variable. Para ello se emplea la dirección de variable. En efecto, la dirección actúa como un intermediario entre la variable y el programa que quiere acceder a ella. En el negocio del espionaje existe un mecanismo análogo: un agente puede dejar su información en un lugar especial (un buzón de correos o un agujero en un árbol) y no tener contacto directo con los otros miembros de la red. Así, si es capturado, la información que el agente pueda revelar bajo tortura es muy pequeña. Podemos decir que el agente tiene sólo acceso indirecto a aquellos a los que proporciona información.

De una forma similar, una instrucción del programa puede referirse indirectamente a una variable, utilizando la dirección de ésta como una especie de buzón de correos para pasar la información.

¿POR QUÉ SE UTILIZAN LOS PUNTEROS?

Los punteros se utilizan en situaciones en las que pasar valores reales es difícil o indeseable (es raro el caso en el que un programa enemigo fuerza a una función a revelar los nombres de las variables utilizadas). Algunas razones para utilizar punteros son:

1. Que una función devuelva más de un valor.
2. Pasar, de una forma más adecuada, *arrays* y cadenas de una función a otra.
3. Manipular *arrays* más fácilmente moviendo los punteros que les apuntan (o que apuntan a algunos de sus elementos), en lugar de mover los propios *arrays*.
4. Crear estructuras complejas de datos, tales como listas indexadas y árboles binarios, donde una estructura de datos puede contener referencias a otras estructuras.
5. Comunicar información sobre la memoria, como con la función *malloc( )*, que informa sobre la localización de memoria libre utilizando un puntero.

De momento vamos a explorar algunos de estos usos de los punteros.Otra razón esgrimida con frecuencia para justificar el uso de los punteros es que la notación con punteros se compila en código más rápido y eficiente que, por ejemplo, la notación de *arrays*. No está claro que éste sea un factor importante para los modernos compiladores; probablemente muchos programadores se acaben *enamorando* de la notación de punteros y abrazarán cualquier excusa para poder usarlos.

Devolución de datos desde las funciones

Comenzaremos nuestro examen sobre punteros descifrando cómo las funciones devuelven valores múltiples a los programas que las llamaron. Ya hemos visto que es posible pasar valores a una función y que ésta devuelva un valor simple al programa, pero ¿qué ocurre cuando quiere pasar más de un valor desde una función al programa? Como las funciones no cuentan con los mecanismos necesarios para realizar esta tarea, debemos confiar en los punteros.

Paso de valores a una función

Antes de mostrar cómo trabajan los punteros, revisemos qué ocurre cuando pasamos valores *a* a una función. A continuación se muestra un ejemplo muy simple que pasa dos valores, los enteros 4 y 7, a una función llamada *gets2( )*:

```
/* valores.c */
/* funciones de prueba con dos valores aceptados
*/
void gets2(int, int);
main( )
{
  int x=4, y= 7;
  gets(x, y);
}

/* gets2( ) */
/* imprime los valores de dos argumentos */
void gets2 (int xx, int yy)
{
  printf("El primero es %d, el segundo es %d", xx,
yy);
}
```

Como todos los objetos de DirectX, lo primero que hay que hacer para poder utilizarlos es obtener una *instancia* por COM mediante

```
LPDIRECTPLAY4   lpDP4 ;

    CoInitialize (NULL) ;

    CoCreateInstance(CLSID_
DirectPlay,
          NULL,
          CLSCTX_ALL,
          IID_IDirectPlay4A,
          (VOID**)&lpDP4);
```

La función CoCreateInstance forma parte de COM y se le pasa como primer parámetro un CLSID (*class* ID) con el número de la clase que identifica a DirectPlay. Como segundo parámetro pasamos NULL, como tercero pasamos CLSCTX_ALL, como cuarto el número que identifica a una instancia IDirectPlay4 y como quinto a un puntero donde guardarla.

Nótese que se ha hecho un *CoInitialize ()* al principio para asegurarnos de que COM se encuentra inicializado, de esta forma, *CocReateInstance* no fallará por este motivo.

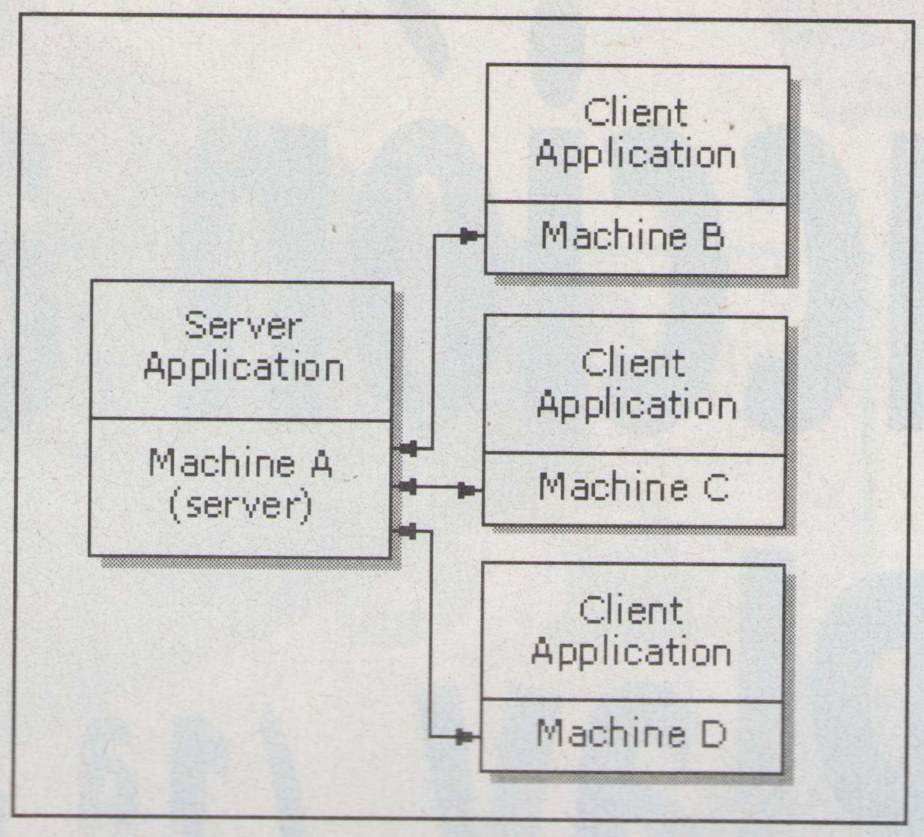
Bien, ahora que ya tenemos un IDirectPlay4, debemos enumerar los *Service Providers* que se encuentran instalados en nuestra maquina. Para ellos, hacemos

```
lpDP4 -> EnumConnections(&g_
AppGUID,
      EnumServiceProviders,
      miHWnd,
      DPCONNECTION_DIRECTPLAY);
```

Como primer parámetro, le pasamos nuestro GUID de aplicación. Si no tenemos uno, podemos generarlo con la utilidad GUIDGen que se proporciona con el paquete Microsoft Visual Studio o pasar NULL. El segundo parámetro indica la dirección de una función *callback* para recibir la enumeración, el tercero un context parameter (en este caso quiero pasarle el *handle* de una ventana List Box para que ponga el nombre del *service provider* en él), y al cuarto le paso siempre DPCONNECTION_DIRECTPLAY.

La función callback en detalle tiene la siguiente sintáxis:

```
BOOL FAR PASCAL
EnumServiceProviders (LPCGUID
pguidSP,
      VOID* pConnection,
      DWORD dwConnectionSize,
      LPCDPNAME pName,
      DWORD dwFlags,
      VOID* pvContext)
```



DirectPlay es capaz de administrar varios modelos de comunicación. Destacan el Client/Servidor y arquitectura distribuida.

```
{
    //Variables locales
    HRESULT hr;
    LPDIRECTPLAY4 lpDirectPlay4 =
NULL;
    VOID* pConnectionBuffer = NULL;
    HWND hWndListBox =
(HWND)pvContext;
    LRESULT iIndex;

    // Creamos un IDirectPlay
    CoCreateInstance
(CLSID_DirectPlay,
          NULL,
          CLSCTX_ALL,
          IID_IDirectPlay4A,
          (VOID**)&pDP4 )º;

    // Comprobamos que la cone-
xión con el Service Provider está dis-
ponible inicializándola y luego destru-
yéndola.
    if (lpDirectPlay4-
>InitializeConnection (pConnection,
0) != DP_OK) return FALSE ;

    //Como ya no necesitamos más
el lpDP4 temporal creado, lo libera-
mos (no confundir con el del principio
del artículo)
    lpDirectPlay4 -> Release () ;

    //Ponemos nombre del Service
Provider en el List Box
    iIndex = SendMessage
(hWndListBox, LB_ADDSTRING, 0,
(LPARAM)pName->lpszShortNameA);
    if(iIndex == CB_ERR) return
FALSE;

    //Reservamos memoria para el
buffer de conexión con el Service
Provider
    pConnectionBuffer = new BYTE[
dwConnectionSize ];
    if(pConnectionBuffer == NULL)
return FALSE;

    // Guardamos GUID en varia-
ble local
    memcpy (pConnectionBuffer,
pConnection, dwConnectionSize);

    //Colocamos GUID en List Box
    SendMessage(hWndListBox,
LB_SETITEMDATA, iIndex,
(LPARAM)pConnectionBuffer);

    return TRUE; // Keep enumera-
ting}
  }
```

Una vez que el usuario elija el proovedor de servicios que desea utilizar (TCP/IP, modem, direct cable, etc...), inicializaremos la conexión con *InitializeConnection()* como se hizo anteriormente.

A continuación, podemos crear una sesión de la siguiente forma:

```
DPSESSIONDESC2 dpsd ;

    memset ( &dpsd, 0,
sizeof(dpsd));
    dpsd.dwSize = sizeof(dpsd);
    dpsd.dwFlags =
DPSESSION_DIRECTPLAYPROTOCOL;
    dpsd.guidApplication =
g_AppGUID;
    strcpy (dpsd.lpszSessionNameA,
"misesion") ;
    dpsd.dwMaxPlayers = 32;
    lpDP4 _ Open (&dpsd, DPO-
PEN_CREATE) ;
```

Básicamente, se rellena una estructura del tipo DPSESSIONDESC2 con el GUID de nuestra aplicación, nombre de sesión y una serie de *flags*. Obsérvese que se utilizará el *Direct Play Protocol*, lo que permite utilizar mensajes garantizados aún en el caso de estar en UDP y IPX. Esto es así porque se reenvían cada cierto tiempo. Otra ventaja de este flag es el aprovechamiento inteligente del ancho de banda o tiempo de respuesta (*throttling*).

Más abajo se especifica el máximo número de jugadores que admitirá la sesión, que serán 32, pero con un valor de 0 podemos hacer que no exista límite de participantes.

Una vez que hemos creado la sesión con IDirectPlay4: Open (), debemos añadirle un *server player*. Para ello utilizaremos el método IDirectPlay4: CreatePlayer () de la siguiente forma:

```
HRESULT CreatePlayer(
    LPDPID lpidPlayer,
    LPDPNAME lpPlayerName,
    HANDLE hEvent,
    LPVOID lpData,
    DWORD dwDataSize,
    DWORD dwFlags
  );
```

Como primer parámetro, pasamos un puntero a una variable que recibirá su ID después de la

llamada. El segundo especifica un puntero a su nombre, el cuarto indica un evento que será señalado cada vez que reciba un mensaje y como sexto pasaremos la constante DPPLAYER_SERVERPLAYER, retornando DP_OK si todo fue correcto.

Bien, si estuviésemos en un modelo Cliente / Servidor, habríamos creado ya éste último. Nos faltaría crear un *player* cliente uniéndolo a la sesión. Conviene enumerar todas las sesiones existentes y preguntar al usuario/a cuál quiere unirse. Para ello, contamos con el método IDirectPlay4: EnumSessions (). Su uso puede verse en el siguiente ejemplo:

```
//Enumera todas las sesiones
LPDPSESSIONDESC2  dpsd ;

memset (&dpsd, 0, sizeof(dpsd));
dpsd.dwSize = sizeof(dpsd);
dpsd.guidApplication =
g_AppGUID;

lpDP -> EnumSessions (&dpsd,
10000, EnumeraSesiones, NULL,
DPENUMSESSIONS_ALL);
```

Como vemos, después de rellenar una estructura DPSESSIONDESC2 con el GUID de nuestro programa y su tamaño, se la pasamos a dicho método, junto con el *time-out* (10 segundos aquí), la dirección de memoria de una función *callback* y un *flag* que le indica que enumere todas las sesiones, admitan o no más jugadores.

Aquí podemos ver en detalle el *callback*:

```
BOOL FAR PASCAL
EnumeraSesiones (LPCDPSESSION-
DESC2 pdpsd, DWORD* pdwTimeout,
DWORD dwFlags, VOID* pvContext)
{
    // Si pasó el timeout, para la
enumeracion
    if (dwFlags & DPESC_TIME-
DOUT) return FALSE;

    // Se ha encontrado una sesión
correcta. Imprime por pantalla su
nombre, número de jugadores actuales y número de jugadores máximos.
    printf("%s (%d/%d)", pdpsd-
>lpszSessionNameA, pdpsd-
>dwCurrentPlayers, pdpsd-
>dwMaxPlayers);

    //Continúa la enumeración
    return TRUE;
}
```

Una vez que el usuario haya escogido en qué sesión quiere entrar, lo unimos a ella con:

```
LPDPSESSIONDESC2 dpsd ;

memset (&dpsd, 0, sizeof(dpsd));
dpsd.dwSize = sizeof(dpsd);
dpsd.guidApplication =
g_AppGUID;
dpsd.guidInstance = pSession-
>guidSession;
lpDP4 _ Open (&dpsd, DPO-
PEN_JOIN) ;
```

Nótese que hemos de indicar el GUID de la sesión a la que queremos unirnos.

El paso siguiente será dar al nuevo jugador de alta. De nuevo, volveremos a utilizar el método IDirectDraw4: CreatePlayer (), tal y como lo hicimos anteriormente.

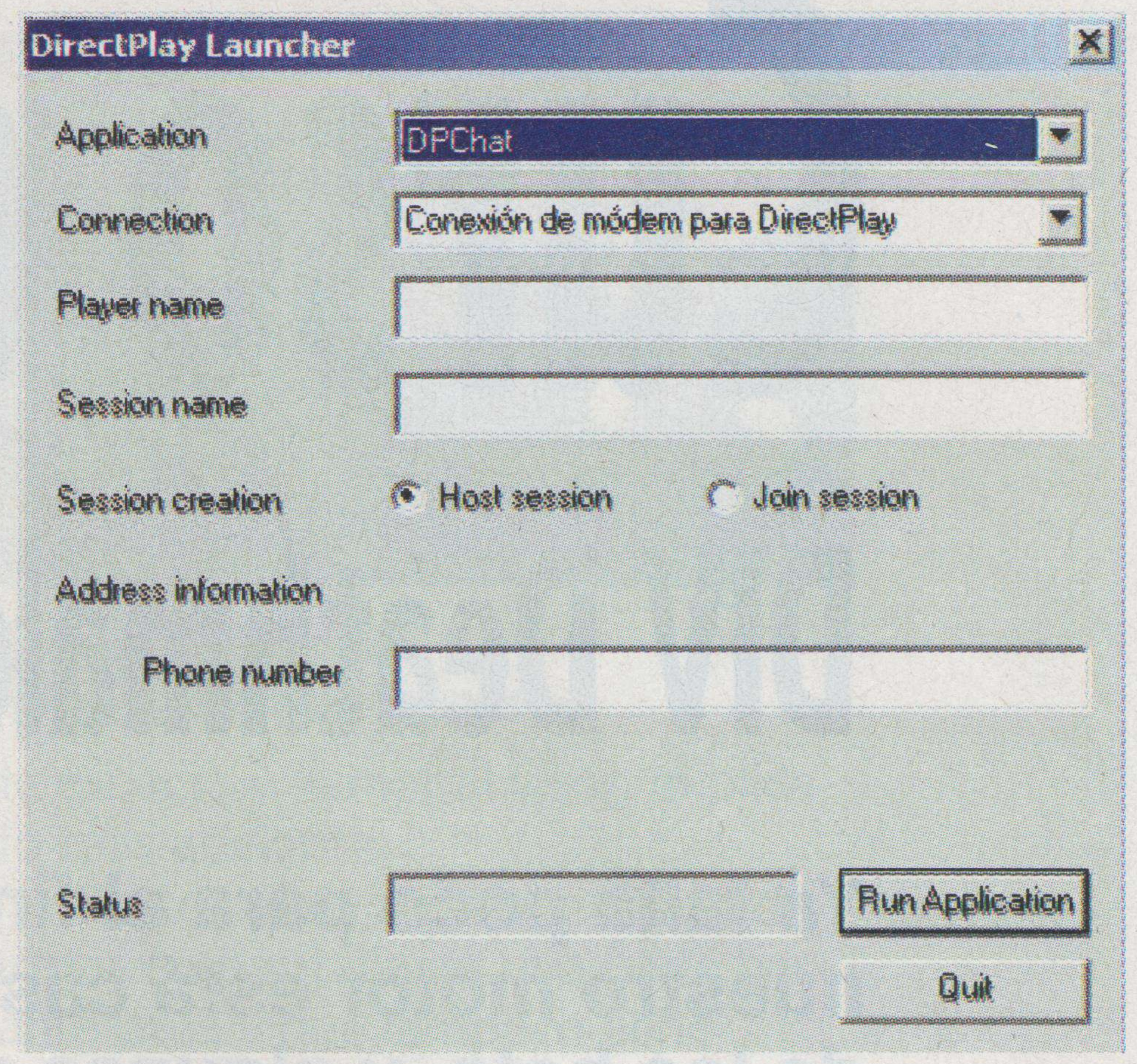
Ahora ya podemos enviar mensajes de uno a otro mediante IDirectPlay4: Send ()

```
HRESULT Send(
  DPID idFrom,
  DPID idTo,
  DWORD dwFlags,
  LPVOID lpData,
  DWORD dwDataSize
);
```

Los dos primeros parámetros especifican el ID del remitente y destinatario. Como *flags* podemos especificar:

- DPSEND_GUARANTEED: Si queremos garantizar la llegada del mismo. No importa que estemos utilizando un protocolo UPD o IPX. Como activamos el DPSESSION_DIRECTPLAYPROTOCOL al crear la sesión, aún tratándose de un sistema no garantizado, se asegurará la recepción y envío de mensajes.
- DPSEND_ENCRYPTED: Podemos cifrar los mensajes, para lo cual, al crear la sesión, debimos especificar un *flag* indicando que se trata de una *secure session*.
- DPSEND_SIGNED: Por si no fuera suficiente con una sesión segura, DirectPlay nos brinda la posibilidad de firmar digitalmente los envíos, para lo que tendremos que tener instalado en nuestra máquina algún mecanismo de certificado.

A la hora de recibir mensajes se desaconseja el uso en bucle de IDirectPlay4: Receive (), siendo mucho más eficiente un sistema de *threads* pasivos, debiendo utilizar en muchos casos mecanismos de memoria compartida, semáforos y señales. El funcionamiento es el siguiente: un hilo despierta por la llegada de una señal externa que activa un evento. Es ese momento podemos leer el mensaje entrante de la cola con




```
HRESULT Receive(
  LPDPID lpidFrom,
  LPDPID lpidTo,
  DWORD dwFlags,
  LPVOID lpData,
  LPDWORD lpdwDataSize
);
```

Los dos primeros parámetros indicarán el remitente y destinatario. Como *flags* podemos especificar:

- DPRECEIVE_ALL - Coge el primer mensaje de la cola y lo borra de la misma.
- DPRECEIVE_PEEK - Coge el primer mensaje de la cola sin borrarlo de la misma.
- DPRECEIVE_TOPLAYER - Activa destinatario. Sólo se cogerán los mensajes dirigidos a él, ignorando los demás.
- DPRECEIVE_FROMPLAYER - Activa remitente. Sólo se cogerán los mensajes enviados por el remitente seleccionado, y se ignoran los demás.

Conclusiones

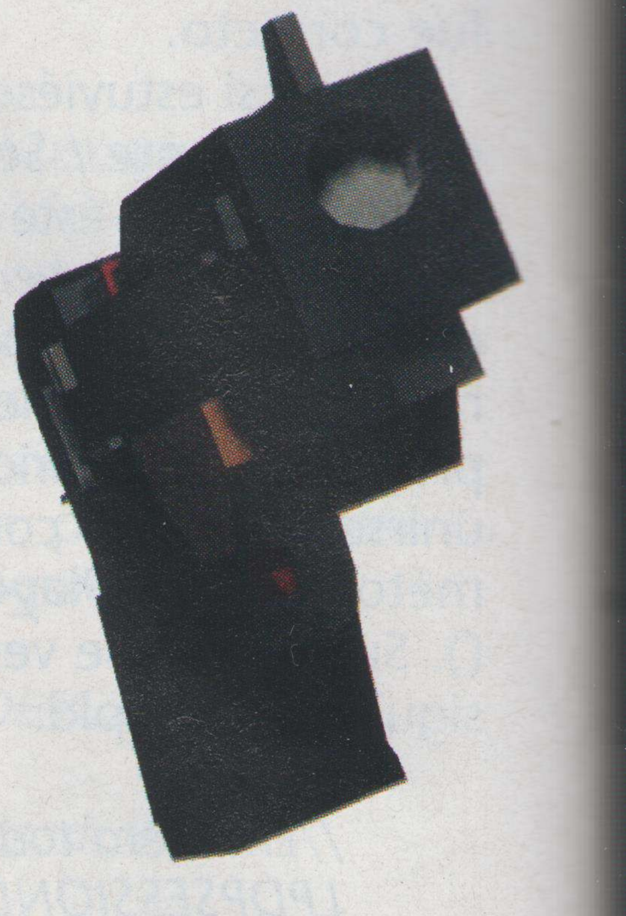
Esto no ha hecho más que empezar. Con DirectPlay podemos hacer infinidad de aplicaciones: desde el más simple programa de *chat* hasta el sistema de red del juego más complicado, pero debido a la intención meramente introductoria de este artículo, no hemos querido profundizar más. En próximas entregas veremos más detalladamente su funcionamiento, añadiremos como ejemplo un pequeño programa y analizaremos algunas características avanzadas, tales como el manejo de grupos o la creación de *lobbies* para servidores en Internet.

De todas formas, si tenéis alguna duda, queréis hacer algún comentario o simplemente intercambiar ideas, podéis encontrarme en *santyhammer@latinmail.com.*

Santiago Orgaz

Curso de juegos en 3D

DIV Deathmaker (V)



Ya falta poco para el final. En este número, nuestro motor está casi al completo. Salvo la IA de los bots, y algunas partes como son los disparos/explosiones y los daños en el jugador y bots, todo lo demás está listo.

Tras haber acabado completamente con la "interfaz" de los menús y, prácticamente todo, salvo el motor del juego (la parte 3D), podemos meternos en profundidad en lo que es la programación pura y dura de juegos. Hasta ahora no había una clara diferencia entre la línea que separa la programación orientada a juegos y la de los programas, pero como notaréis, en esta última la programación toma una orientación distinta.

Lo primero que se puede observar en nuestro nuevo PRG (el cual ha crecido de forma espectacular) es que muchas partes han sufrido cambios, en especial, la declaración de variables. Como todos sabréis, al empezar un programa se tiene una base mental sobre la estructura general que va a tener, pero a medida que se avanza en el programa esta base que se fijó va a ir modificando y ampliando. Con DDM no ha podido ser de otro modo; son muchas las modificaciones como para enumerarlas todas, pero por ejemplo:

Algunos datos se guardaban en variables tipo PRIVATE. Al comprobar más tarde que era necesario acceder a esas variables desde otro proceso, se han tenido que cambiar algunas a GLOBAL, o LOCAL... también pasa lo mismo con variables que se guardaban en tablas tipo WORD que más adelante, al observar que era necesario acceder a su OFFSET (desde un write_int, por ejemplo) se han tenido que pasar a variables "sueltas".

El portal se hace más grande

Nuestra función *portalmuerte()* ha aumentado (y bastante) en código. Antes de nada, quiero comentar que HABÍA UN ERROR en el PRG de la anterior Divmanía ("mea culpa", lo siento), exactamente en la asignación de registros de PLAYER[7], parte que consideré como complicada y quiso la fortuna que justo ahí se diera un BUG. En este número ya está corregido.

En el anterior número la función *portalmuerte()* cargaba y preparaba perfectamente el modo8 y, de hecho lo sigue haciendo, aunque se pueden apreciar los cambios a partir del fin del bucle principal de juego: ahí comienza un largo proceso de DESCARGA de todo lo que había en memoria y no era necesario. De todos modos, hay un problema de memoria descrito más adelante.

Otra novedad en *portalmuerte()* es una parte que gestiona unas variables llamadas "mensaje1", "mensaje2", etc... explicado en la sección "EL MSJMASTER".

Por último, os quiero recordar que *portalmuerte()* TODAVÍA no guarda los resultados de una partida, por razones obvias. Todavía no se puede "matar" a nadie, porque las armas no disparan perfectamente (aunque sí hacen la animación y llaman al proceso disparo) y no está implementado el reconocimiento de colisiones y daño.

El MSJMASTER

Este nombre tan curioso es el que alberga uno de nuestros procesos, además uno muy importante, se trata del "Maestro de los mensajes", es decir, el proceso que gestiona la "cola" (o pila) de mensajes, añadiendo mensajes nuevos, reteniéndolos por un tiempo y eliminando los antiguos cuando se sobrepasa el límite.

En realidad, el *msjmaster()* funciona en conjunto con *portalmuerte()*, puesto que es esta última función la que se encarga de ir borrando mensajes con el tiempo y mostrarlos a cada FRAME. *Msjmaster()* simplemente se encarga de los mensajes nuevos.

Cuando queremos meter un mensaje en la cola llamamos a *msjmaster*, por ejemplo, con la siguiente sintaxis: *msjmaster("Este es un mensaje de prueba")* y *msjmaster* comprueba si hay un "slot" o mensaje libre en orden y lo introduce allí; si no encuentra ningún mensaje libre (o mejor dicho, línea de mensaje), corre todos un lugar hacia arriba y lo coloca el primero.

Como podéis observar, ya están implementados todos los mensajes de recogida de objetos y armas. Faltan los mensajes que indican las muertes y asesinatos del jugador y los bots.

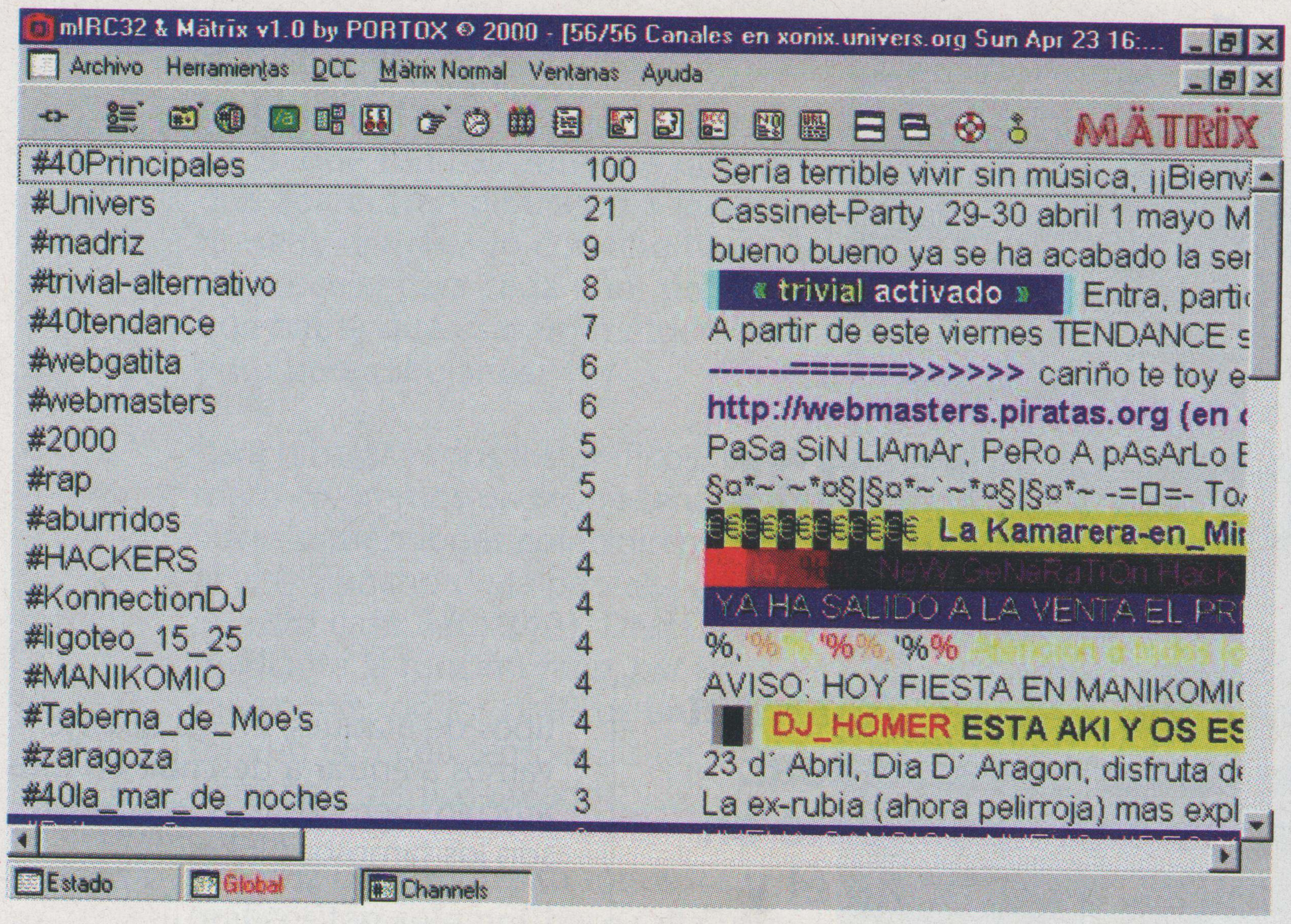
Contrólame a ese jugador

Llegamos al proceso más "escabroso", por decirlo de algún modo, del programa. Algunos pueden pensar que en la parte de la IA de los bots es necesario extenderse más que en el proceso del jugador. En cierto modo así es, sin embargo, no hay tanta diferencia, porque lo que hacen los bots está de cierta manera "escondido"; no necesita representación en pantalla, mientras que lo que hace el jugador sí. Por supuesto, los bots también tienen que estar un tiempo cambiando de arma y recargando (aunque en algunos juegos parece ser que cambian instantáneamente de arma).

El proceso del jugador es muy largo como para explicarlo en su totalidad. Hay varias partes que siguen una estructura similar, así que nos basaremos en la composición general del proceso. Por supuesto hay que recordar que todavía está incompleto, aunque no falta mucho.

Empezamos con un esquema básico del proceso jugador actual que podéis ver en la Tabla 1.

Como veis, son varias partes,

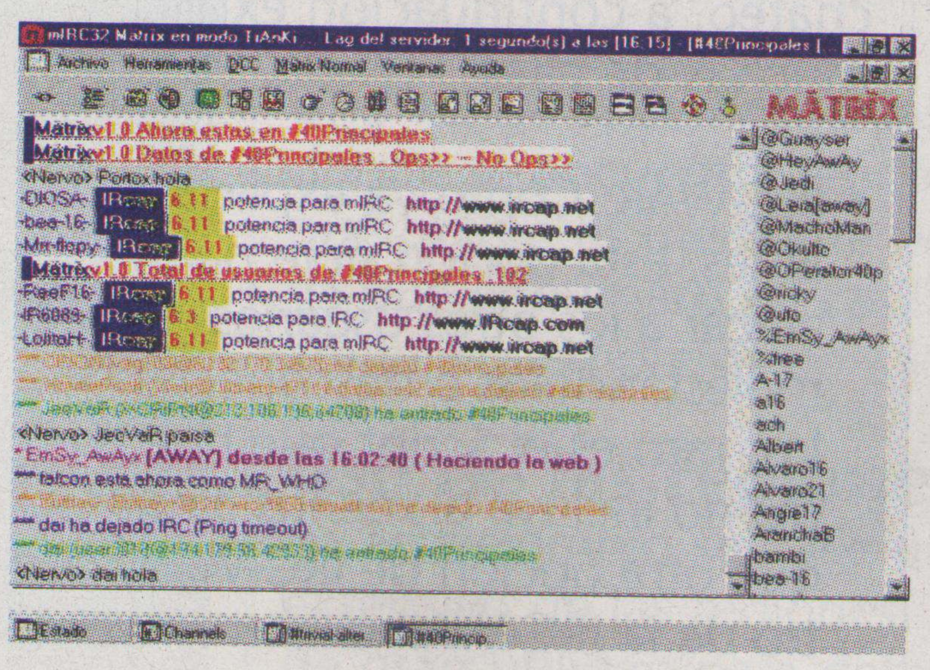


seamos OP, como kickear o banear, pero estas opciones se ven incrementadas en los Scripts, ya que incluyen opciones como hacer OP´s masivos o kicks y bans masivos, o kicks con frases divertidas. También podemos activar protecciones para que el Script kickee automáticamente a aquellos usuarios del IRC que estén molestando, de esta manera no tendremos que estar vigilando quién molesta o no para echarlo, el Script lo hará automáticamente por nosotros así podemos hacer que el Script kickee automáticamente a quien escriba palabrotas, quien tenga clones, quien flodee el canal escribiendo muchas frases seguidas, quien escriba con mayúsculas y muchas otras opciones mas.

Otras opciones

Pero no os penséis que todas las opciones extras de los Scripts están dedicadas íntegramente a atacar y defenderse de los ataques de otros usuarios, como os mostraremos a continuación existen una inmensa cantidad de opciones que no están pensadas mas que para amenizar el rato que pasemos en el IRC:

- TEXTOS ESPECIALES: Con este nombre designamos las opciones que tienen muchos Scripts para poder escribir lo que deseemos con letras y colores especiales, así, podemos escribir textos con letras gigantes, con los colores de nuestro equipo favorito, con sombras, con marcos, en bloques , y con muchos otros estilos.
- MENSAJES ESPECIALES: En este apartado se incluyen las opciones que tienen muchos Scripts para poder dedicar mensajes especiales de bienvenida, despedida, de amor, de amistad, de acciones y otros muchos más, a un nick.
- DIBUJOS: Los dibujos son archivos TXT , formados mediante puntos, guiones, barras, letras u otros signos, que debido a la unión de estos, forman diferentes dibujos que pueden ser en blanco y negro o en color y que se muestran al canal, existen multitud de dibujos, algunos muy curiosos, que van desde objetos hasta animales o personas.
- JUEGOS: Existen distintos juegos para entretenernos con otros usuarios del IRC mientras chateamos, nos podemos encontrar desde el famoso tetris, hasta el ahorcado, el cara y cruz (muy útil para echar algo a suertes) e incluso existen juegos que nos afectan de verdad mientras chateamos, aunque para ello quien lo active debe ser OP, como la maquina tragaperras con la cual podemos ganarnos desde un OP hasta un kickeo, un baneo o un Nuke y la famosa ruleta rusa, con la cual, al que se le dispare el arma será kickeado automáticamente.



- SONIDOS: Otra opción interesante es la referente a los sonidos, muchos scripts incluyen opciones especiales referentes al tema, como archivos MIDI o MP3 que podemos escuchar mientras chateamos, además podemos reproducirlos a todo el canal y no sólo a nosotros, para que también lo oigan los demás usuarios del IRC y por si esto no fuera suficiente también se puede disponer de la opción de escuchar la radio en directo a través de Internet, para que no nos podamos aburrir en ningún momento mientras estemos en el IRC.

Existen distintos juegos para entretenernos con otros usuarios del IRC mientras chateamos

- RELOJ: El reloj es una opción muy corriente, aunque existen varios tipos, los hay que simplemente te muestran la hora y los hay que nos dicen el tiempo exacto de conexión e incluso algunos el gasto en pesetas que llevamos efectuado desde que nos conectamos, muy útiles para que la factura de telefónica no nos sorprenda a final de mes.
- ENCRIPTACIÓN: La encriptación es un método utilizado en el IRC para transmitir mensajes codificados y sirve para que nadie mas que los que tengan la clave de descodificación puedan leerlo, existen muchos métodos de encriptación con códigos diferentes, de manera que si escribimos un mensaje y lo encriptamos en uno de esos códigos, las personas que no posean el código con el que hemos encriptado o cifrado el mensaje solo verán aparecer en pantalla una serie de extraños signos. Para que os hagáis una idea, la palabra DIV-MANIA encriptada en diferentes códigos tendría este aspecto:

Código ScriPtteR´s v2.1:
ïÆ•¢©¥Æ©
Código AZZ-CODE: *ª"^¨%ª¨
Código Legendary CODE: ~§

¿Qué es el LAG?

Es la cantidad de tiempo en segundos entre lo que un usuario escribe y otro u otros leen lo escrito. Es decir, si hay dos segundos de lag entre 'A' y 'B' entonces lo que A escriba, B no lo va a ver hasta dentro de dos segundos después. Por lo tanto cuanto menos lag haya entre los usuarios, más ágil y rápida será la comunicación entre ellos.

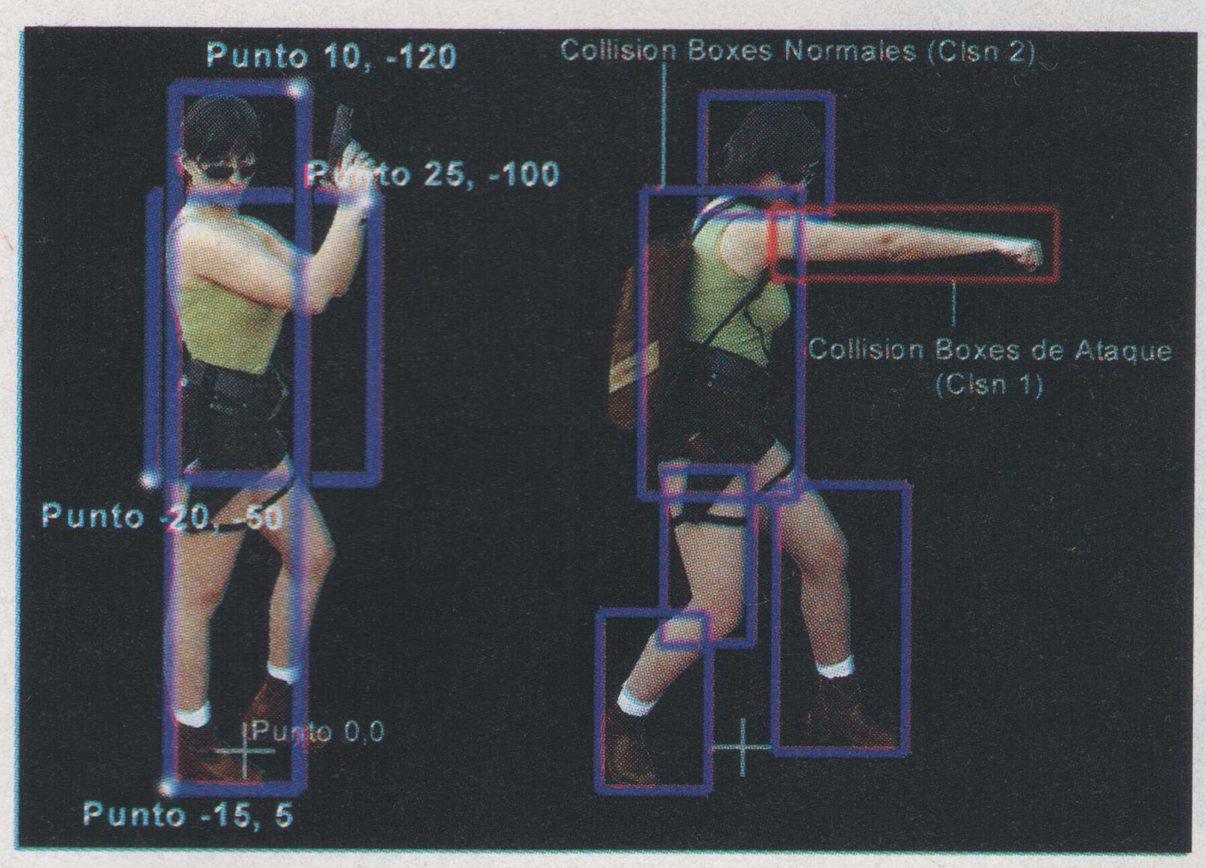


Fig. 3: Collision Boxes normales y de ataque.

un fichero de texto. Encontraréis las instrucciones para hacer esto en el *readme.txt* de las *Tools*. Otra opción que seguramente os resultará más sencilla es usar el mencionado programa *Mugen Character Maker*, que hace la conversión mediante un práctico *interface* visual. (ver Fig. 2).

Nota importante: Los luchadores está limitados a una paleta de 8 bits, y todos los sprites deben tener la misma paleta. Esto no debe resultar una gran limitación, y de hecho con DIV ocurre lo mismo.

• ***luchador.snd***: Es el fichero de sonidos, que se crea de manera muy similar al *.sff*, pero partiendo de ficheros *.wav* y utilizando el programa Sndmaker de las *Tools*. Hay que decir que este fichero no es imprescindible, pero si no lo incluimos nuestro luchador no tendrá sonidos.

Pueden participar hasta cuatro luchadores simultáneos, sin contar los helpers

• ***luchador1.act, luchador2.act, etc.***: Estos ficheros también son opcionales, y contienen las distintas paletas que tendrá el luchador en función de con qué botón lo seleccionemos en el juego. Se pueden crear con Photoshop o cualquier otro programa que pueda salvar paletas con extensión *.act*. Si no los incluimos el luchador sólo tendrá una paleta, la del fichero *.sff*.

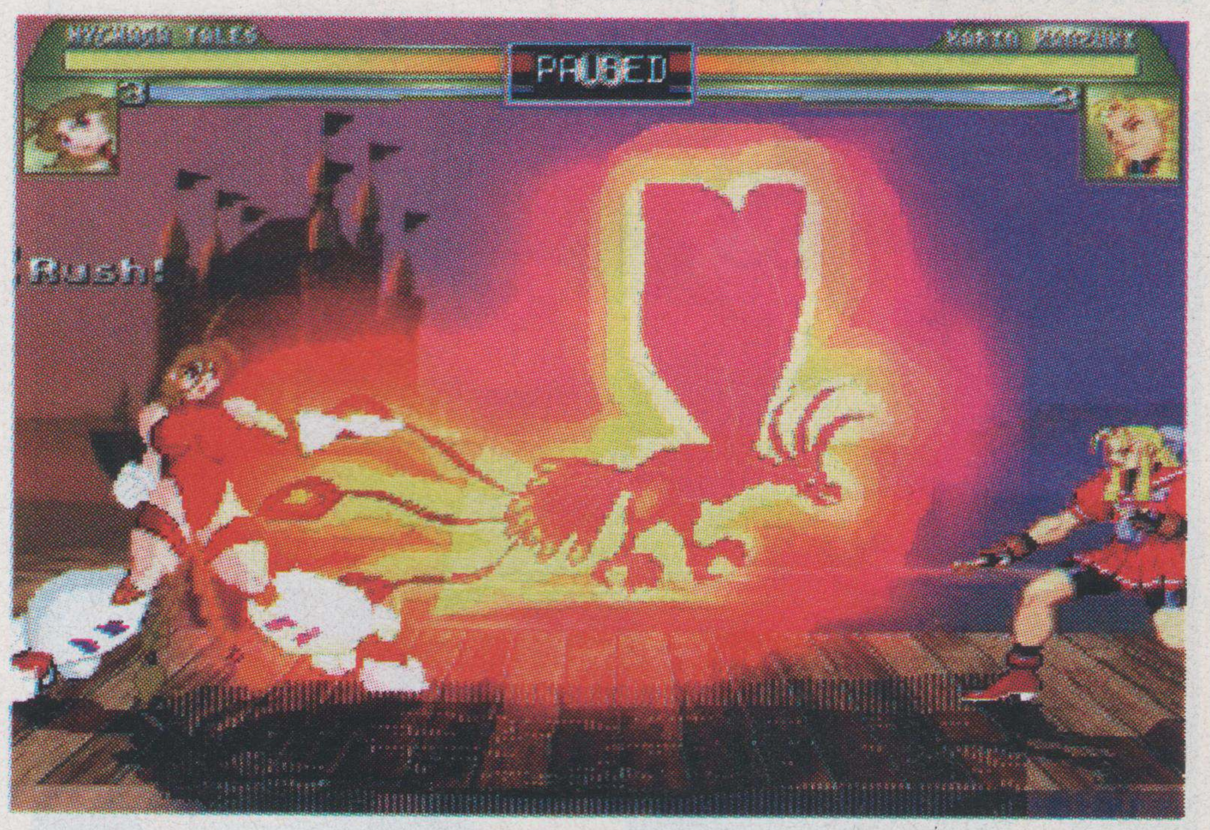


Fig. 4: Los efectos de transparencia permiten crear impresionantes ataques.

Ahora veremos los ficheros de texto que marcan el comportamiento del luchador. Para crearlos sólo necesitáis un editor de texto. Lo mejor es que modifiquéis los que vienen en *player.zip* y les cambiéis el nombre por el de vuestro personaje.

• ***luchador.air***: Aquí se describen dos cosas: las animaciones del personaje y sus *collision boxes,* que no son otra cosa que rectángulos que describen su forma. Los *boxes* de tipo 1 definen las zonas con las que el luchador puede golpear, y los de tipo 2 las zonas donde puede ser golpeado (ver Fig. 3).

Cada animación y sus correspondientes *boxes* se incluyen en una acción. Una acción es una lista de sprites (procedentes del fichero *.sff*) con sus *collision boxes* correspondientes. Veamos esto con un ejemplo:

```
;Posición Inicial
[Begin Action 0]
Clsn2Default: 2
 Clsn2[0] = -15,5,10,-120
 Clsn2[1] = -20,-50,25,-100
0,1,0,0,30
0,2,0,0,30
0,3,0,0,30
0,2,0,0,30
```

Esta es una de las acciones de un fichero *.air*. La primera línea es un comentario, que el programa ignorará al ir precedido del punto y coma. La segunda es la definición de la acción, que nos dice que su número de identificación es el 0. Después tenemos el número de *collision boxes* de tipo 1. El nombre *Default* indica que éstos son validos para toda la acción (si pusiera *Clsn2: 2* sólo

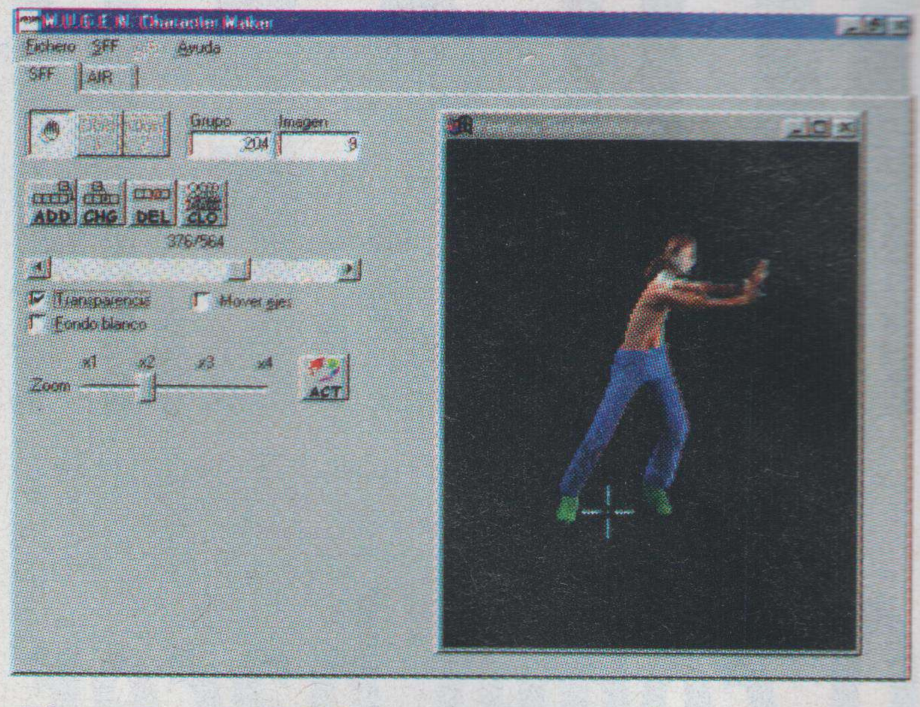


Fig. 2: M.C.M. es un excelente programa para crear personajes.

serían válidos para el primer *sprite*). Después están las coordenadas de cada *box*, que se corresponden con la imagen de la izquierda de la Fig. 3. Y finalmente tenemos la lista de sprites, cada uno con sus valores: el grupo y el número de sprite (descritos en el archivo *.sff*), sus coordenadas (normalmente 0,0) y el número de *frames* que durará el *sprite*. Tened en cuenta que el juego funciona a 60 *frames* por segundo, así que esta acción duraría dos segundos.

Otra cosa que debéis saber es que el programa tiene unas acciones obligatorias, que debéis poner siempre, y que describen las acciones básicas del luchador. Así, la acción 0 es la del luchador quieto, la 20 la de caminar, etc. Además hay unos *sprites* obligatorios en el fichero *.sff*. Podéis encontrar una lista completa de las acciones obligatorias en *mugen\docs\air.txt*, y de los *sprites* en *sprite.txt*.

Las animaciones del archivo *.air* nos permiten hacer muchas más cosas de las que hemos visto hasta

Cómo crear los sprites

Lo primero que uno necesita para crear un luchador son sus *sprites*, es decir todas las imágenes que describen al luchador en todos sus movimientos posibles. Hay muchas maneras de conseguir estos *sprites*; podéis crearlos desde cero con un programa de dibujo o con uno de modelado tridimensional, o también podéis tomar fotografías vuestras con una cámara digital. Pero la manera más fácil y popular es la de "robar" los *sprites* de un juego de lucha. A este proceso se le llama *ripping*.

Para hacer esta captura necesitáis mucha paciencia, pues se trata en esencia de hacer capturas de imagen del juego. Después deberemos limpiar de cada imagen todo lo que no sea el luchador, dejando sólo el color 0 de fondo.

Lo más normal para hacer esto es usar un emulador, pues hay muy pocos juegos de lucha para PC, mientras que la PlayStation o la Neo Geo tienen muchísimos. En cualquier caso, recordad que debéis tener los derechos de uso de los juegos (es decir, los juegos originales) para usarlos con el emulador.

Sin duda el mejor emulador para hacer esto es el NeorageX, que aparte de emular estupendamente la Neo Geo, tiene una genial herramienta para eliminar de la imagen todos los elementos que no pertenezcan al luchador. (Ver Fig. 7).

Direcciones de Interés

L*http://www.elecbyte.com*: Los creadores de M.U.G.E.N. Aquí encontraréis la última versión de este programa y de las *M.U.G.E.N. Tools*.

http://come.to/testp: Los testeadores oficiales han creado un juego completo usando M.U.G.E.N. Lo tenéis también en el CD de DIV Manía. Sus luchadores son los mejores para aprender el funcionamiento del programa.

http://go.to/mugen4ever: La página de fans más completa. Aquí están casi todos los luchadores que existen para M.U.G.E.N.

http://teleline.terra.es/personal/moi.ses: En la Página de Sés encontraréis el fantástico programa *M.U.G.E.N. Character Maker*. Muy recomendable si queréis crear un luchador.

http://perso.club-internet.fr/moah_/KOF91/: La competencia. Ésta es la página oficial de KOF 91 (Ver el cuadro correspondiente)

ahora, como por ejemplo invertir la imagen, hacer bucles o aplicar efectos de transparencia, una característica muy potente que podéis ver en la Fig. 4.

-luchador.cmd: Está dividido en dos partes. En la primera hay una lista de comandos, que no son más que los botones que hay que pulsar para hacer un ataque. Su descripción es similar a las acciones pero más sencilla. Veamos, por ejemplo, como se ejecutaría un *Hadoken*:

; Bola de Fuego
[Command]
name = Hadoken
command = ~D, DF, F, a
time= 30

Tras el comentario y la definición, viene el nombre del movimiento. Es importante que anotéis el nombre pues lo necesitaréis para activarlo después. Después viene la descripción del comando: soltar abajo (*~D)*, pulsar abajo y delante (la diagonal), pulsar delante y a (un botón de puñetazo). Si queremos hacer un movimiento en el que haya que mantener pulsado un botón, por ejemplo atrás dos segundos, se haría así: *command= ~120$B, F, a* .Es decir, soltar atrás después de 120 *frames*, o 2 segundos, pulsar delante y a. Finalmente tenemos *time= 30*, que indica el tiempo que tenemos para ejecutar la acción; si tardamos más no se activará.



Fig. 6: El luchador de la izquierda no tiene icono ni portrait.

La segunda parte del *.cmd* es más compleja y define cuándo podemos hacer cada movimiento para que, por ejemplo, no podamos hacer una patada aérea estando en tierra. Explicaré cómo funciona más tarde, pues su uso es muy similar al del siguiente archivo.

-luchador.cns: Describe los movimientos que el personaje puede realizar. Este es el último fichero que necesitamos, y sin duda alguna el más complicado. Llegar a dominarlo es casi imposible, especialmente porque la documentación referente a su uso está incompleta. Sin embargo, los movimientos básicos no son complicados de hacer.

El fichero *.cmd* es similar al *.air*, y así como éste se organiza en acciones, el *.cmd* es una lista de estados. Por ejemplo, al empezar el combate el luchador está en el estado de *Intro*, para después pasar al estado inicial; si pulsamos entonces arriba, el luchador pasará al estado de prepararse para saltar, y después al de saltar arriba, etc.

Por suerte para nosotros, la mayoría de estados ya están hechos, de manera que no tendremos que definir cómo caminar, saltar o agacharse. Los únicos estados importantes por definir son los ataques del luchador. Por eso veremos cómo se definen.

Un estado empieza por su definición, seguido de unos parámetros básicos. Después vienen los controladores. Los controladores son la parte más potente de todo el programa, y también la más complicada. Permiten al luchador hacer casi cualquier cosa, desde crear proyectiles a cambiar los colores del escenario o crear un luchador ayudante (Ver Fig. 5). Para activar estas acciones se utilizan los *triggers*, o gatillos. Clarificaré esto con un ejemplo:



Fig. 5: Ejemplo de Helpers: Megaman y Galford con sus mascotas.

;Puñetazo Medio y
[Statedef 200]
type = S
movetype= A
ctrl = 0
anim = 200

[State 200, 1]
type = HitDef
trigger1 = AnimElem = 6
attr = S, NA
damage = 20
animtype = Hard

[State 200, 2]
type = ChangeState
trigger1 = AnimTime = 0
trigger2 = Time > 20
value = 0
ctrl = 1

Esto es más complicado que lo visto hasta ahora, y de hecho se puede complicar mucho más. Veámoslo por partes:

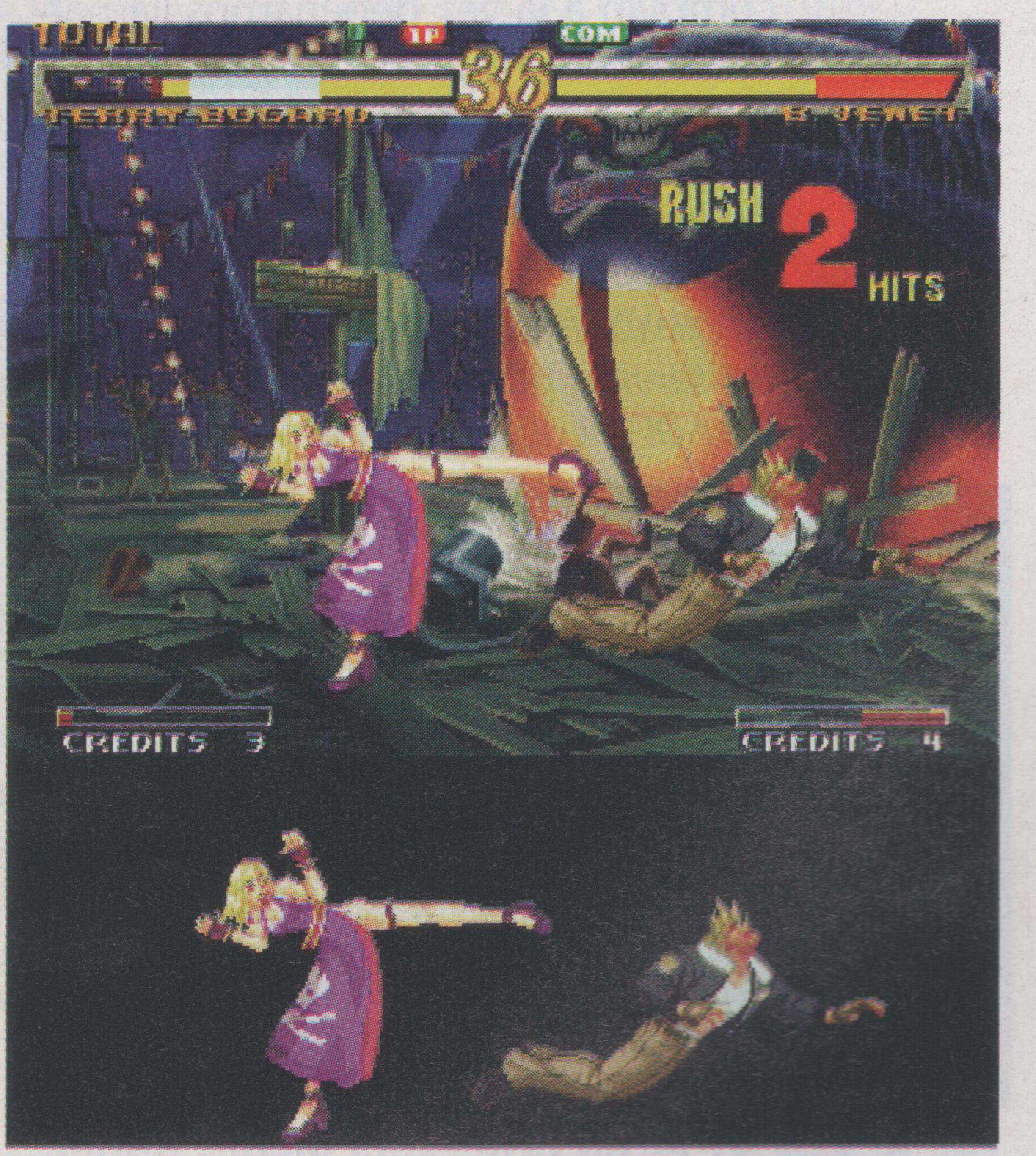


Fig. 7: Con NeoRageX el ripping de sprites es muy sencillo.

Primero viene el comentario y la definición. Después los parámetros básicos: tipo de estado o *type* (S- De pie, C- Agachado, A- En el aire, L- Tumbado en el suelo), tipo de movimiento (A- Ataque, D- Defensa, I- Todos los demás), si el jugador tiene o no control (0 para que no tenga control, 1 para que pueda interrumpir el estado y pasar a otro al pulsar los botones), y finalmente el número de animación. Para que el estado funcione correctamente deberemos haber definido esta animación en el *.air*, con sus *collision boxes* de ataque. Después viene un controlador (Cada estado puede tener tantos controladores como queramos). Tiene su definición y su tipo, en este caso definir un impacto o *HitDef*. Después viene un *trigger*, que es la condición que debe cumplirse para activarlo, en este caso que estemos en el sexto *sprite* de la animación. A continuación vienen los parámetros del estado. *Hitdef* es un estado muy complejo y puede tener muchos parámetros, por lo que os aconsejo que miréis detenidamente el ejemplo que viene en *player.cns*. En este caso tenemos los atributos (S- de pie, NA- Ataque normal), la energía que restará el impacto a nuestro contrincante y el tipo de animación que tendrá éste al recibirlo. (*Hard*, *Medium* o *Low*)

La flexibilidad del engine permite hacer supers con todo tipo de efectos

Finalmente tenemos otro controlador, que cambia al estado inicial y proporciona control al jugador cuando la animación llega a su fin o el tiempo es superior a 20. Para definir esta condición alternativa, se define un *trigger2*. Si, en cambio, quisiéramos que las dos condiciones se cumplan a la vez deberíamos llamarlo también *trigger1*. Para más información sobre los *triggers*, que sí están bien documentados, podéis consultar *mugen\docs\trigger.txt*.

Sin embargo, sólo con esto el estado no serviría para nada, porque no le hemos dicho al programa cuándo activarlo. Falta introducir el cambio a ese estado en la segunda parte del archivo *.cmd* (Atención: esto va en el archivo *.cmd*, no en el *.cns*):

```
;Puñetazo Medio y
[State -1]
type = ChangeState
value = 200
trigger1 = command = y
trigger1 = command != holddown
trigger1 = statetype = S
trigger1 = ctrl = 1
```

El estado -1 es un estado que siempre está activo sea cual sea el estado del luchador, y sus controladores sólo pueden ser de tipo *ChangeState*. Aquí vemos el que corresponde al golpe que acabamos de definir: el luchador pasará al estado 200 siempre que se cumplan simultáneamente cuatro condiciones: que hagamos el comando llamado "y", que no pulsemos abajo, que el luchador esté de pie y que tenga control.

Hasta aquí llega la descripción de los archivos. Pero ya hemos visto que el programa es complejo, y los archivos largos y numerosos. La pregunta que os debéis hacer es: ¿por dónde empiezo? Pues bien, aquí tenéis unos consejos para empezar.

Cómo empezar un luchador

Lo primero que necesitáis son los *sprites* del luchador (Ver el cuadro correspondiente). A continuación debéis descomprimir *player.zip* y renombrar los archivos con el nombre de vuestro luchador. Después deberéis modificar el *.def*. Seguidamente debéis crear el fichero *.sff* y el *.air* con las animaciones obligatorias, descritas en *mugen\docs\air.txt*. Esto es sencillo y gratificante, pues desde la animación 0 cada vez que queráis podéis cargar el programa y ver a vuestro luchador en acción.

Además, para que aparezca la imagen de vuestro luchador en la pantalla de selección de personajes tendréis que introducir su icono (imagen 9000,0 del *.sff*) y su *portrait* (9000,1). Si no lo hacéis al seleccionarlo su imagen estará en blanco, como veis en la fig. 6.

Y finalmente, debéis crear las animaciones de los ataques, sus entradas y comandos correspondientes en el *.cmd* y sus estados en el *.cns*. Y con esto ya tendríamos un luchador completamente funcional. Obviamente podemos hacerle muchas más cosas, como introducir sonidos, animaciones de introducción y final de *round*, ataques especiales, etc. Y por supuesto podéis crear su escenario con música, elementos animados, *scrolls*... Quizá podamos ver todo esto en un futuro artículo. Pero una vez entendidas las bases, todo lo demás llegará con la práctica. Espero que este artículo os haya servido de ayuda y ver pronto vuestros luchadores en la red.

Héctor Zapata
zapata@ncsa.es



Fig. 8: KOF 91 es un programa menos potente que M.U.G.E.N., pero más sencillo.

El otro M.U.G.E.N.: King of Fighters 91

En Internet existen otros programas que permiten personalizar en mayor o menor medida juegos de lucha. El más popular después de M.U.G.E.N. es un interesante juego para Windows llamado KOF 91. (Ver Fig. 8).

La filosofía de trabajo de este programa es muy distinta. Todos los luchadores tienen las mismas características y los mismos movimientos, y lo que los diferencia son los propios *sprites*. De manera que más que crear un luchador, lo que hacemos es substituir los *sprites* de un luchador existente por los nuestros, poniendo las imágenes *.pcx* en su directorio correspondiente.

Las ventajas son obvias: no hay complicados archivos de texto que modificar ni archivos de imagen y sonido que crear. Por lo tanto, crear un luchador puede ser cuestión de minutos una vez tenemos los *sprites*. Además posee un editor de luchadores muy fácil de usar. Pero ya os podéis imaginar los inconvenientes: no tenemos apenas flexibilidad, los luchadores tienen muchos menos movimientos y la jugabilidad en general es mucho peor. Pero también tiene interesantes añadidos, como por ejemplo un efecto de *zoom* muy conseguido. Además al ser para Windows es más fácil de usar.

Este programa está dirigido a aquellos que no tengáis ganas de complicaros la vida aprendiendo el funcionamiento de M.U.G.E.N., o para los que quieran hacer un luchador rápidamente. Pero probablemente os quedaréis con ganas de más, pues el programa es bastante limitado en cuanto a creación de luchadores.

MP3 y WINAMP



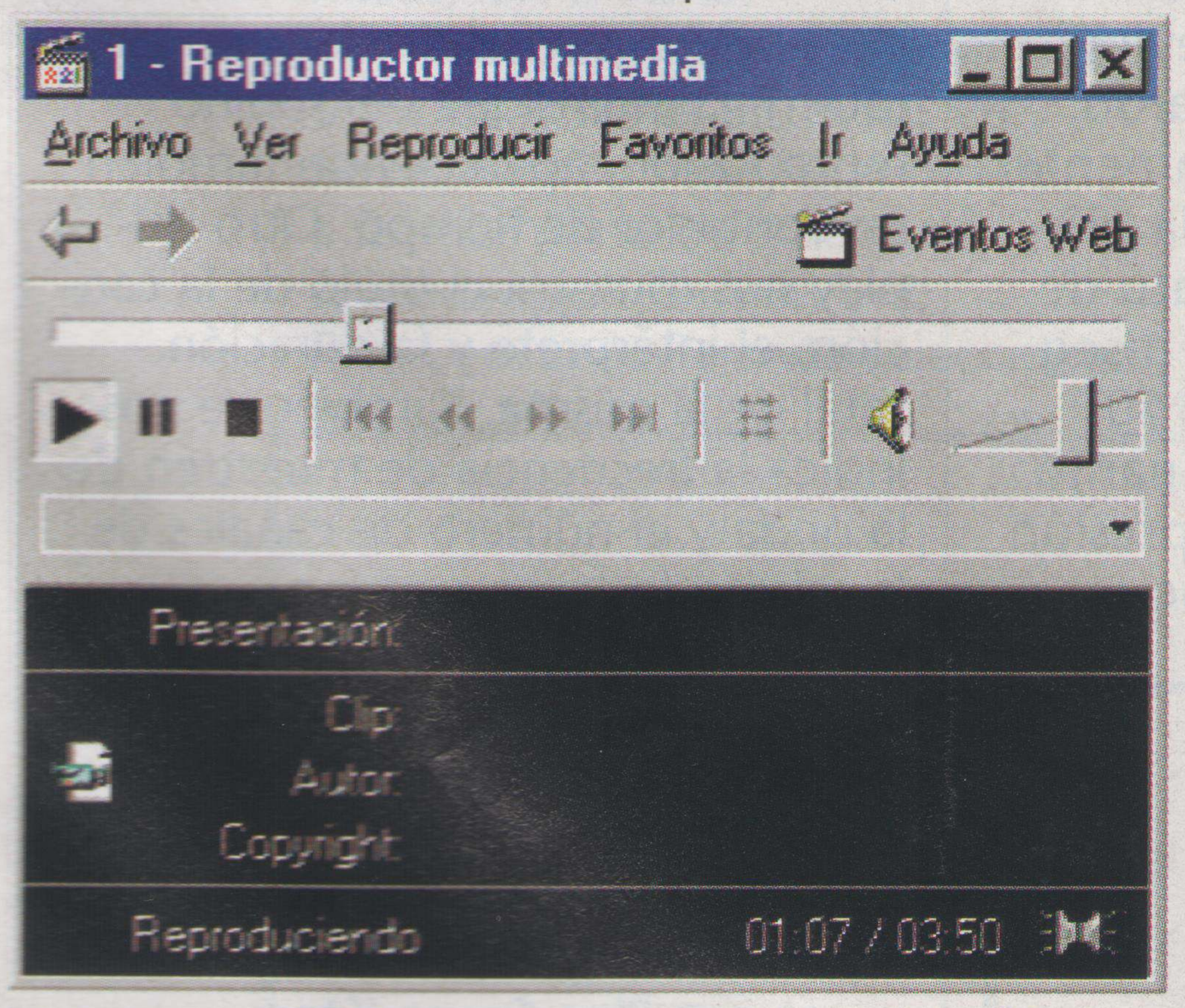
La revolución del audio en la red

Veremos con detalle en este número de DIVmanía un formato sonoro que ha hecho cambiar radicalmente todo este submundo de la informática, abriendo un nuevo y amplio abanico de posibilidades. Nos referimos al flamante y casi mágico formato MP3.

El avance de la informática en general, se caracteriza por tener un crecimiento exponencial y todo lo relacionado con el sonido digital no se queda atrás: hace 10 años, un computador apenas era capaz de emitir algún que otro tímido pitido. Sin embargo, desde hace unos pocos años, los ordenadores son capaces de grabar, e incluso reproducir sonido extraído de fuentes externas, con una calidad idéntica a la original. El único inconveniente que se tenía hasta hace bien poco, era la gran capacidad de disco duro que se requería para almacenar sonido con una calidad aceptable, lo cual hacía imposible, inevitablemente, el intercambio de este tipo de archivos, tan enormes, a través de Internet.

Antes de nada, ¿qué es MP3?

MP3 no es ni más ni menos que un formato de compresión de audio.



El formato MP3 es un standard reconocido por Windows.

Antes de entrar en más detalles deberíamos intentar comprender lo que es un formato de compresión: pues bien, por compresión se designa a la acción de reducir de algún modo el volumen o espacio ocupado por un archivo. Podríamos dividir los formatos de compresión de ficheros en dos tipos:

- Los que comprimen uno o más archivos en un solo fichero, que únicamente podrá ser descifrado haciendo uso del programa descompresor correspondiente. Este tipo de formato de compresión es el que se utiliza para hacer backups, o para trasladar información en una cantidad arbitrariamente grande, de un ordenador a otro. Nunca podremos ejecutar o leer un archivo contenido dentro del fichero comprimido de esta manera, si antes no lo descomprimimos. Esta es la principal diferencia de este tipo de formatos de compresión con respecto a los del segundo tipo. Ejemplos de este tipo de formatos de compresión son *.ZIP, *.RAR, *.ARJ, *.AIN, *.ACE, etc.
- Los que permiten que el archivo comprimido sea leído y utilizado por programas concretos, realizando la descompresión en tiempo real, si es necesario. Es el caso de JPEG, MPEG o el que nos ocupa, MP3.

Las siglas MP3 representan en realidad la frase MPEG Audio Layer-3. Esto significa que MP3 es la parte sonora del formato de compresión de vídeo MPEG, que es el standard desarrollado por el MOVING PICTURE EXPERT GROUP para este efecto. Es por eso que, a partir de determinada versión de Windows 95, el propio sistema operativo provee de aplicaciones para reproducir archivos de vídeo en formato MPEG, y por consiguiente, también para reproducir archivos sonoros MP3.

¿Cuáles son las ventajas de MP3?

Como ya se ha podido vislumbrar en la introducción del artículo, la principal ventaja de MP3 es su poder de compresión: haciendo uso de un compresor MP3 seremos capaces de comprimir un fichero de tipo WAV de manera que pase a ocupar una décima, o incluso una doceava parte de lo que ocupaba originalmente, sin alterar perceptiblemente la calidad sonora del fichero original.

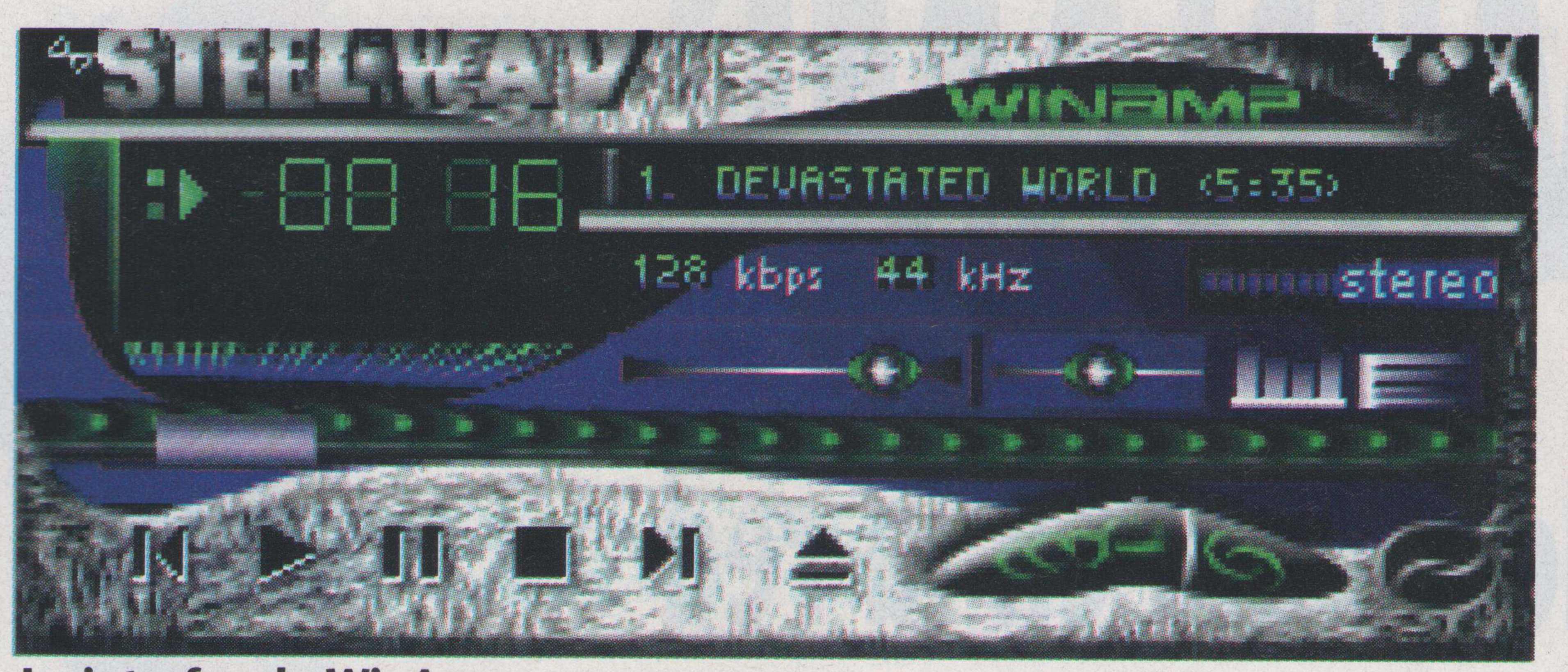
La compresión MP3 se basa en la supresión de frecuencias no perceptibles por el oído humano.

Gracias a esto, ahora podemos transportar a través de Internet canciones completas, sin que esto suponga una espera eterna, y el consiguiente gasto telefónico. Veamos un ejemplo con cifras: Imaginemos que queremos enviar a través del correo electrónico una de nuestras canciones favoritas a un destinatario cualquiera. Pongamos que la duración de la misma es de 5 minutos y que queremos enviarla con calidad CD, es decir, sampleada a 44100 Hz, en estéreo, y con 16 bits por muestra. Realizamos las siguientes operaciones:

- Samplear con una frecuencia de 44100 Hz significa que se origi-



WinAmp es un reproductor de sonido que acepta MP3.



La interfaz de WinAmp.

narán 44100 valores cada segundo durante el proceso. Tenemos que multiplicar ese valor por dos, puesto que, al ser estéreo, tenemos dos canales por los cuales estamos recibiendo información.

- Debemos multiplicar de nuevo el valor obtenido por dos, ya que cada muestra, por ser de 16 bits, ocupara 2 bytes de memoria en nuestro disco duro.
- Por último, multiplicamos por 300 (segundos), ya que habíamos dicho que la canción duraba 5 minutos.
- Obtenemos por tanto, que la canción ocupa la friolera de 50 Mbytes.
- Si ahora multiplicamos los 50 Mbytes por 8 para obtener el número de bits de información que representan, y dividimos por los 28.800 baudios de un módem de calidad media, y por los 60 segundos que tiene un minuto, se obtiene que tardaríamos alrededor de 4 horas (245 minutos para ser exactos) para poder enviar este archivo a través de la red.

MP3 es un formato de compresión de audio muy eficaz y potente.

En caso de pasar a formato MP3 el WAV en discordia, nos hubiera bastado con unos 20 minutos, lo cual parece bastante más razonable.

¿Cómo funciona MP3?

Si hay algún lector que no haya escuchado todavía un archivo MP3, seguramente estará bastante escéptico, y le costará creer que no se pierda ninguna calidad de sonido si se logra un índice de reducción de espacio tan alto. Por otra parte, los lectores que hayan comprobado por sí mismos las cualidades de MP3, estarán pensando que es cosa de magia. Pues no se trata de ningún fenómeno paranormal. Es más bien una especie de truco, digno del mismísimo Copperfield, que consigue engañarnos con una eficacia sorprendente. La codificación MP3 se denomina perceptual, y esto viene a significar que se basa en el conocimiento de las características de nuestra percepción auditiva.

El proceso de compresión, que es bastante complejo, podría resumirse de la siguiente manera: se transforman las muestras o samples en valores de frecuencia aplicando una transformada rápida de Fourier (Fast Fourier Transform) y posteriormente se suprimen aquellas frecuencias que no son perceptibles por el oído humano, mediante la aplicación de un modelo psicoacústico (psicoacustic model). Es por eso que no notamos diferencia al reproducir los archivos MP3, cuando en realidad ha sido modificada considerablemente la onda original.

Limitaciones de MP3

Aunque MP3 es un potentísimo formato de compresión, tiene un inconveniente: como la descodificación o descompresión se realiza en tiempo real, esto toma un tiempo, lo cual origina, inevitablemente, un pequeño retardo. Este retardo es aproximadamente de 50 mseg en el caso de la primera versión de este formato de compresión (Layer 1), 100 mseg en la segunda (Layer 2) y 150 mseg si se trata de MP3. Cuanto mayor es el índice de compresión mayor es el retardo. Esto no representa un problema para el transporte masivo de sonido por Internet (los retardos citados son despreciables), pero esto puede suponer un problema para el uso de MP3 en el campo de la telefonía, ya que el retardo acumulado puede llegar a hacer las comunicaciones bastante torpes. Para aplicaciones telefónicas es preferible reducir directamente la frecuencia de muestreo, reduciéndose así la calidad del sonido hasta el límite en el que las palabras siguen siendo comprensibles (formato PCM).

Otra limitación de MP3 es que sus reproductores requieren de cierta potencia de CPU, ya que realizan, como ya hemos dicho, descompresión en tiempo real. Sería imposible reproducir MP3 con plena calidad en un 486 DX2, por ejemplo.

Rendimiento de cada una de las capas sonoras MPEG

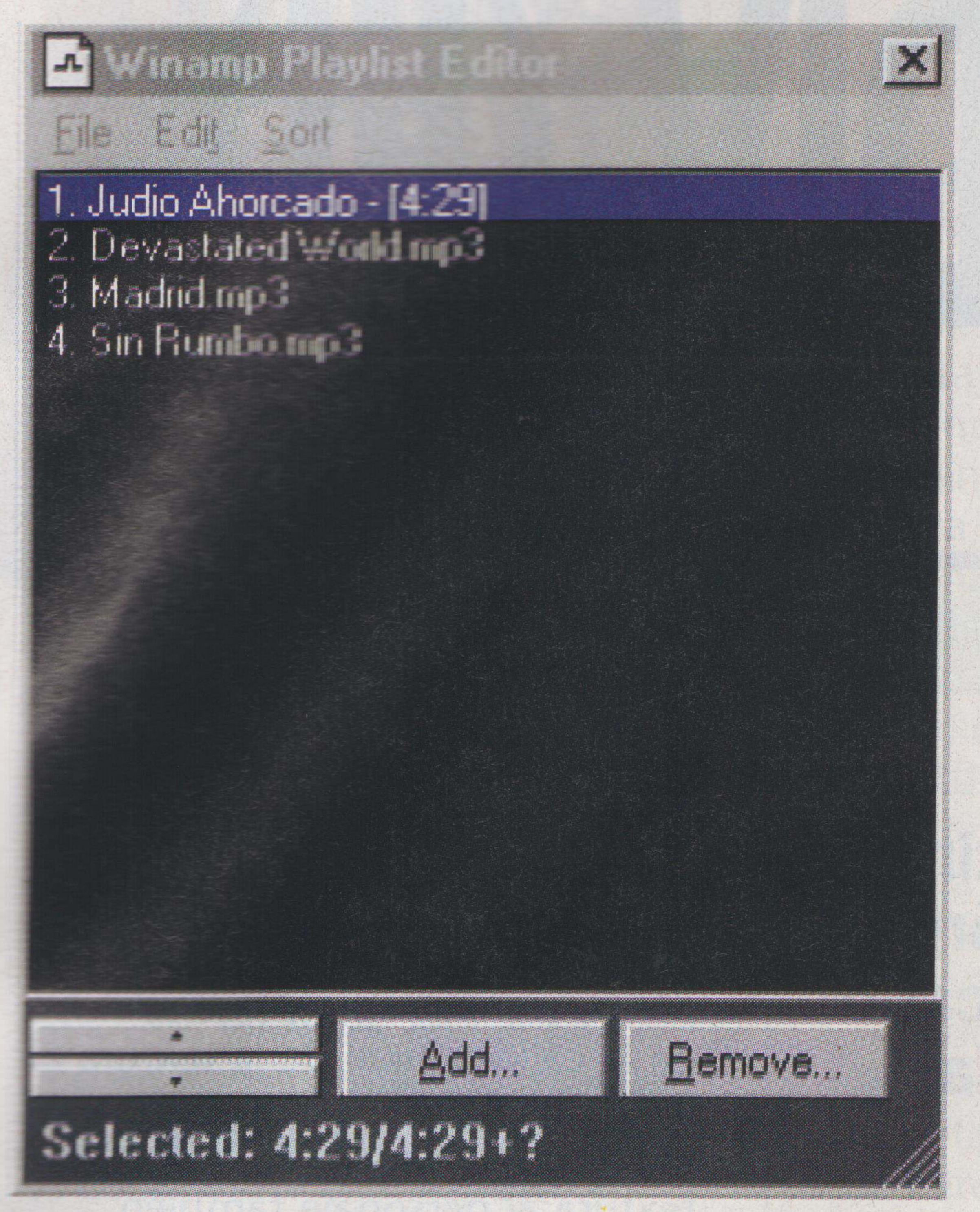
Capa	Factor de compresión	Retardo
Layer 1	1:4	< 50 mseg.
Layer 2	1:6 - 1:8	100 mseg.

Programas que pueden reproducir MP3: WinAmp

El reproductor multimedia de Microsoft puede reproducir MP3, pero no nos ofrece la posibilidad de modificar las características del sonido, ni de manejar un grupo grande de ficheros. Todo esto y más lo encontramos en el que es, posiblemente, el programa de reproducción de MP3 más extendido. Nos referimos a WinAmp.

El interfaz de WinAmp

WinAmp es un programa reproductor de numerosos formatos de audio entre los que se encuentra MP3. Su interfaz emula al frontal de un típico aparato reproductor de Compact Disc, permitiendo cambiar la apariencia del mismo con las diferentes pieles o "skins" que podremos adquirir en cualquiera de los sitios web relacionados. La interfaz tiene un display que representa, en forma de barras volumétricas o bien en forma de onda (esto se cambia haciendo clic en el display con el ratón), la intensidad del sonido reproducido, apareciendo también un contador de tiempo. Otro display nos muestra el título de la canción, el intérprete y la duración total de la misma. También se nos indica la frecuencia de muestreo de la onda y el número de canales de la misma (estéreo o mono). Podremos variar el volumen y la panorámica con dos controles de desplazamiento horizontal. Así mismo se dispone de todos los controles típicos de manejo de un reproductor de CD convencional, como son Play, Pause, Stop, FastForward, Rewind, Canción siguiente, Canción anterior,



WinAmp permite crear listas ilimitadas de reproducción.



El ecualizador gráfico de WinAmp.

Introducir nuevos archivos (es decir, cargarlos desde disco) e incluso los usuales Shuffle (Random) y Repeat.

Todos estos elementos pueden variar ligeramente dependiendo del skin que estemos utilizando, pero básicamente, dispondremos de los mismos controles.

Las opciones de WinAmp

Haciendo "clic" en la parte superior izquierda de la ventana de WinAmp se nos despliega un menú con diferentes opciones.

WinAmp es el reproductor de formato MP3 más generalmente extendido.

Son las siguientes:

- WinAmp: muestra información sobre el programa en general, como los créditos de realización del mismo, el precio que hay que pagar para registrarlo (WinAmp es Shareware, claro), las mejoras de la versión en concreto o direcciones URL (web) de páginas relacionadas con el tema.
- Play File: Permite seleccionar uno o varios ficheros ubicados en nuestro disco duro, que pasarán a formar parte de la lista de reproducción de WinAmp.
- Play Location: Permite seleccionar un fichero ubicado en una máquina remota, a través de Internet, para reproducirlo desde nuestro ordenador.
- View File Info: Nos da información sobre el fichero que actualmente está siendo reproducido (título de la canción, autor, álbum, año de creación, etc).
- Graphical Equalizer: Nos da acceso a un ecualizador gráfico de 10 bandas que oscilan entre 60 Hz y 16 Khz, así como a un control de preamplificación.
- PlayList Editor: Lista de archivos a ser reproducidos. Se nos da la posibilidad de ordenarlos por título, por nombre de fichero o aleatoriamente.
- Options: Esta opción despliega otro menú que nos da acceso, entre otras cosas, a las preferencias del programa y al selector de skins (pieles). Podremos encontrar pieles para todos los gustos: agresivas, clásicas, sobrias, coloristas e incluso temáticas (desde Star Wars a Los Simpsons, pasando por Spiderman). También podremos cambiar otros parámetros como el tamaño de la ventana de WinAmp.
- PlayBack: Estas opciones son accesibles directamente desde la interfaz.
- Visualization: Permite modificar el plug-in preseleccionado. Este plug-in podrá ser un volúmetro u osciloscopio determinado de los que tengamos disponibles en nuestro WinAmp, que será mostrado a pantalla completa. Algunos de ellos son realmente atractivos y podremos activarlos pulsando con el botón derecho del ratón sobre uno de los displays de la interfaz.

Dónde encontrar todo este material

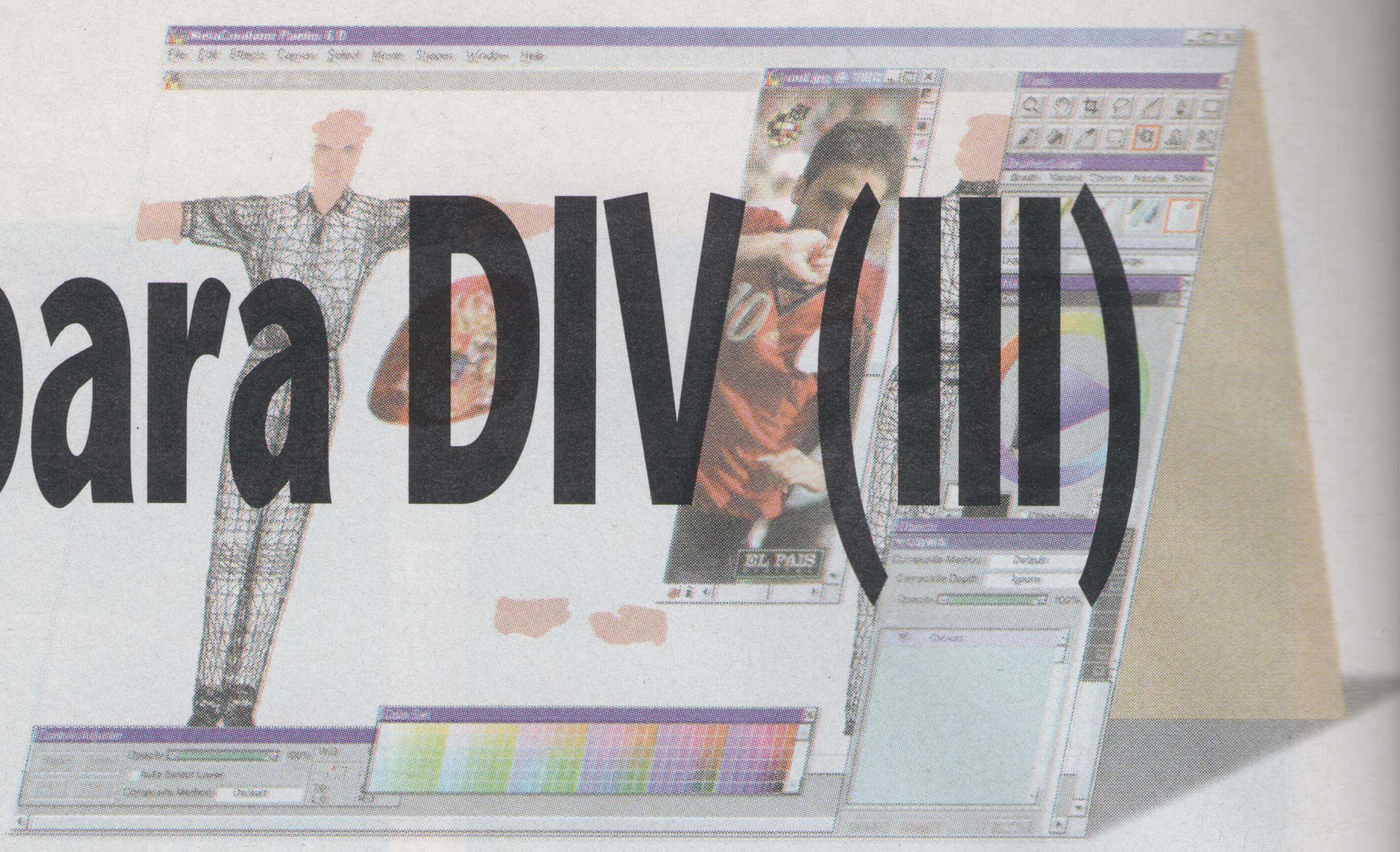
Para encontrar todo tipo de archivos MP3 podremos dirigirnos a páginas como www.mp3.com que nos da acceso a gran cantidad de canciones, escrupulosamente clasificadas por estilos. Podremos tanto bajarlas de la red (download), como escucharlas directamente. En el caso de bajarlas podremos optar por diferentes calidades, para minimizar el tiempo de download, si así lo deseamos. De todas formas es conveniente hacer uso de alguno de los programas destinados a acelerar la transferencia de archivos en Internet, como Netants, ya que, aunque MP3 ahorra mucho tiempo y espacio, los archivos siguen siendo bastante extensos (una media de 4 a 5 Mbytes).

Para buscar música independiente o autoproducida podemos buscar en www.vitaminic.com, que ofrece la posibilidad de dar publicidad y comercializar sus obras en formato MP3, a los músicos tanto profesionales como amateur. Para conseguir la última versión de WinAmp bastará con navegar hasta www.winamp.com.

Sergio Cánovas

Formatos que es capaz de reproducir Winamp

Formato	Extensión
Impulse Tracker Module	*.IT
Extended Module	*.XM
ScreamTracker 3 Module	*.S3M
ScreamTracker 2 Module	*.STM
MultiTracker Module	*.MTM
ProTracker Module	*.MOD
Layer 2 MPEG	*.MP2
Layer 3 MPEG	*.MP3



Posando para DIV (III)

Aplicando texturas

En el artículo anterior estudiamos cómo crear distintas posturas secuenciales que nos diesen la sensación de movimiento. Hoy veremos un punto que ha provocado el mayor número de consultas por vuestra parte: la aplicación de texturas.

Dado el interés que habéis demostrado, en este número comentaremos los métodos para añadir realismo y color a nuestros personajes Poser, aplicando imágenes bitmap a su superficie.

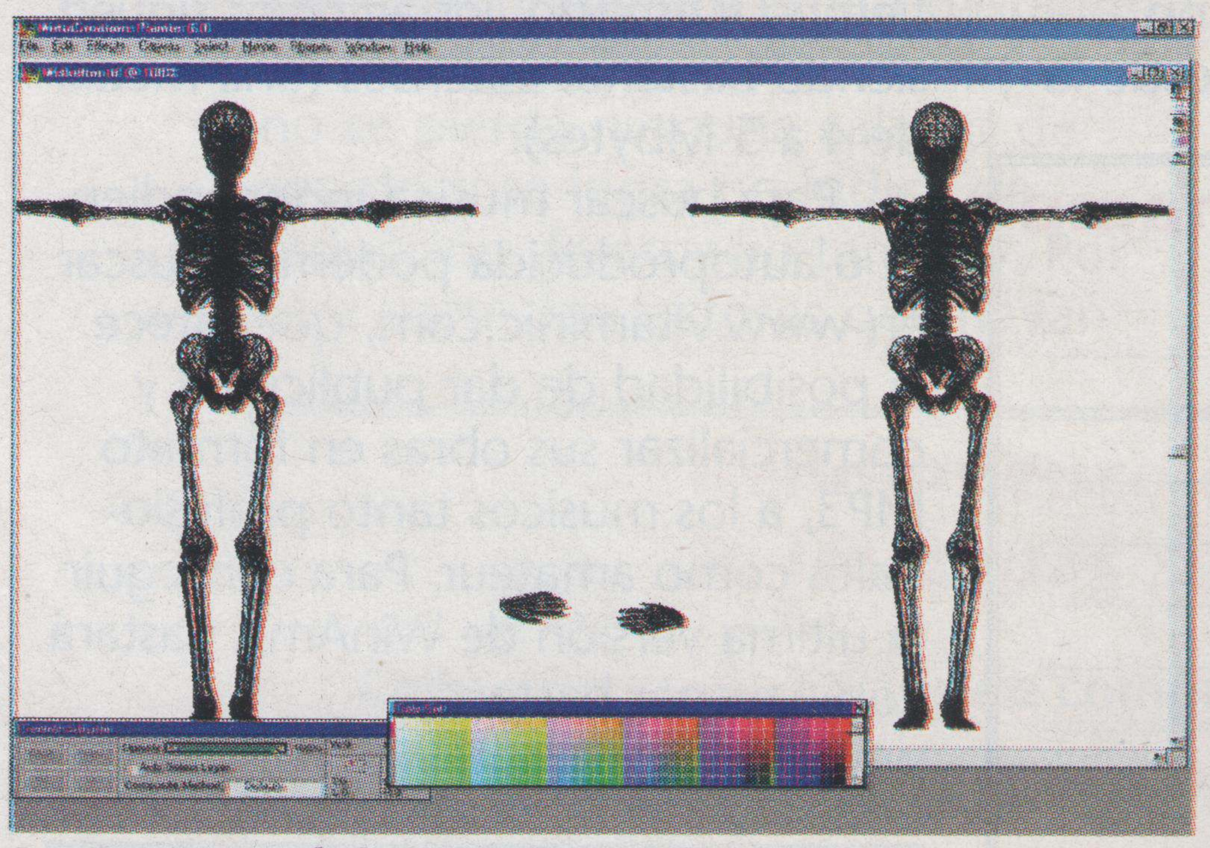
Quizás es el primer paso a la hora de diseñar vuestros personajes del juego, que seguro os traéis entre manos: definir el "aspecto" que tendrá el personaje para luego aplicar el movimiento y animación y posteriormente, crear los sprites correspondientes.

Generalmente, al crear el guión de los juegos, se suele discutir el aspecto que tendrá el personaje: "Detective joven con camiseta blanca y pantalones vaqueros", "Elfo de piel blanca, con chaleco verde y botas rojas", etc.

Pues eso es precisamente lo que haremos a continuación, definir el aspecto de nuestro personaje.

Aplicando texturas

Llegamos al momento que tantas consultas ha provocado en mi buzón: el personalizar nuestro personaje aplicándole los colores y ropas que deseemos. Para ello, como siempre, debemos tener muy claro antes de empezar, las características de nuestro personaje, en definitiva, "crearle una historia": ¿ha luchado cuerpo a cuerpo y tiene cicatrices? ¿su ropa es recién comprada o está llena de barro y polvo? ¿es de piel blanca o está tostado por el sol?



La textura del esqueleto.

También debemos decidir cómo vamos a crear la textura. Vaya esto por delante: cualquier programa de dibujo sirve para realizar las "pieles" de nuestros personajes en Poser. Es decir, cada uno puede utilizar su programa de retoque favorito: Photoshop, Paint Shop, Painter, Paintbrush (que viene por defecto con Windows), e incluso el propio editor de Div.

Obviamente, existen otras opciones más "profesionales" que seguro que todos aquellos que manejáis programas 3D conocéis: me refiero a las aplicaciones de texturizado como Deep Paint 3D o la propia de Metacreations 3D Paint. Estas utilidades te permiten pintar directamente sobre un objeto 3D, como si se estuviera pintando una maqueta o un lienzo y nos muestran en todo momento el resultado final de nuestras pinceladas. Así, sólo habría que cargar en el programa de texturizado el personaje de Poser (guardado antes desde este programa en alguno de los formatos 3D que reconozca vuestro texturizador: 3DS, OBJ, etc...) y aplicarle la textura. La imágen bitmap generada por esa textura debe guardarse después en alguno de los formatos reconocidos por Poser (por ejemplo, BMP).

Para los que utilicéis este método de texturizado (programas *ad-hoc),* os doy una serie de consejos que seguro os son de utilidad:

- Haced que el tamaño del bitmap de la textura sea lo suficientemente grande (1024x1024 pixels, por ejemplo), pues influye en la calidad final de la textura al aplicarla.
- El método de aplicación de la textura al personaje también influye. Si desde un programa 3D escogemos un mapeado cilíndrico, cúbico, plano, etc... para aplicar al objeto Poser antes de texturizar, esto tendrá una influencia decisiva en el resultado final.
- Moved la luz en el programa de texturizado, pues una zona en sombra que creamos que está bien pintada, puede presentar pinceladas mal dadas, trazos sin cubrir, etc...

De todas formas, sabiendo que el presupuesto de casi todos es muy reducido como para permitirnos adquirir una de estas utilidades de aplicación directa de texturas, lo haremos con nuestro programa de retoque gráfico tradicional. Yo utilizaré Painter 6 de Metacreations, porque es el que me ha parecido que se adapta mejor a mi forma de trabajo. Vosotros podéis usar cualquier programa, lo único que se requiere es que sea capaz de trabajar con partes de imagen cortadas y pegadas (el editor de Div y el Paintbrush que viene de serie con Windows pueden hacerlo).

Como ejemplo y dado que cuando estoy escribiendo el artículo quedan pocas fechas para la Eurocopa de fútbol, haremos un pequeño homenaje a la Selección



Los elementos iniciales.



Pongámosle cara a nuestros personajes.

Española creando un personaje que será un jugador de la misma, para un juego de fútbol.

Los datos de partida son sencillos: en cualquier revista, periódico o página Web encontraremos fotos del uniforme de la Selección, así que sólo tenemos que copiar el diseño de la camiseta y pantalón nacional para que nuestro personaje cobre vida. En concreto, yo seleccioné de Internet la imagen que podéis ver a la derecha. De ahí puedo copiar el dibujo de la camiseta y por otra parte, aparece el escudo de la Selección bien claro. Todo esto nos será de gran utilidad en los siguientes pasos.

Por otro lado, para darle un aire más personal, vamos a ponerle al jugador la cara que nos parezca mejor. Si tenéis cámara digital, sólo tenéis que fotografiar a la persona que queráis y pasar la foto al ordenador como JPG. También se puede seleccionar una foto que tengamos y escanearla. El caso es tener una imagen JPG de la cual podamos "cortar y pegar" una cara para añadirla a la textura de nuestro jugador. En los próximos minutos veremos cómo hacerlo.

Las plantillas de texturas de Poser

Cada modelo de persona de Poser (hombre de negocios, niño desnudo, etc...) tiene una forma distinta definida por las arrugas de la vestimenta, calzado, etc... De ahí que la forma de pintar su textura es diferente.

Metacreations nos facilita las cosas dándonos las texturas básicas de cada figura disponible en Poser. Si abrimos el directorio "Templates" en el CD de Poser, veréis que hay una serie de dibujos en formato TIF, que pueden ser abiertos en cualquier programa de dibujo. Abrid por ejemplo, "Mskelton.tif" y podréis observar que lo que ahí se refleja es el dibujo de los polígonos que componen el modelo de esqueleto masculino de Poser, dividido por partes y en la postura básica (con los brazos extendidos). Las manos se encuentran separadas de la figura central y el cuerpo se muestra de frente y de espaldas. Lo que representa la plantilla es un "desenrolle" (un "unwrap" en inglés) del objeto Poser en todos los polígonos que lo componen. Si pintamos directamente sobre esta imagen y la cargamos posteriormente en Poser, adquiere la pintura que hayamos añadido a esta imagen bitmap. Puede sonar complicado, pero ahora veremos que no es así.

Así, cada modelo de Poser lleva su textura correspondiente, sólo hay que mirar el nombre en inglés de cada una para imaginar su correspondencia con los modelos del programa.

Dado que Poser no incluye ningún modelo en pantalón corto y camiseta para nuestro personaje, lo que haremos será pintar la textura de "Hombre vestido informalmente" (Casual man), simulando la camiseta y el pantalón corto de la Selección.

Comencemos por abrir en nuestro programa de dibujo la plantilla "CASMAN.TIF" del directorio "Templates" del Cd Poser. Como podéis comprobar, se trata del modelo de hombre informal dividido en polígonos, se puede reconocer dónde acaban las mangas de la camiseta, la cintura, etc...

Lo normal es comenzar pintando las partes que quedan al descubierto, como los brazos, piernas, cara, etc..., seguir con la vestimenta y acabar con el calzado. Así que procederemos a ponerle cara a nuestro personaje y pintaremos el resto de cuerpo descubierto.

Recortad de la imagen JPG que deseéis la cara de la persona a "retratar" en el personaje. Todos los programas de dibujo tienen una herramienta para seleccionar zonas en el dibujo, así que seleccionad la cara de la foto que os guste y "cortad" la selección (en casi todos los programas suele estar bajo "Edición>Cortar"). Ahora tenéis la cara de vuestra imagen en el portapapeles de Windows y sólo hay que pegarla en la plantilla que hemos abierto CASMAN.TIF. Como podeis ver arriba yo he tenido que disminuir el tamaño de la cara recortada y rotarla un poco para que quede recta.

Pegad la cara y, si como me ha sucedido a mí, no encaja con la de la plantilla por ser muy grande, no estar completamente recta, etc... rotadla y escalarla con vuestro programa de dibujo, hasta dejarla con las dimensiones de la plantilla de nuestro personaje. Una vez encajada la cara, seleccio-



Pintando los colores básicos.

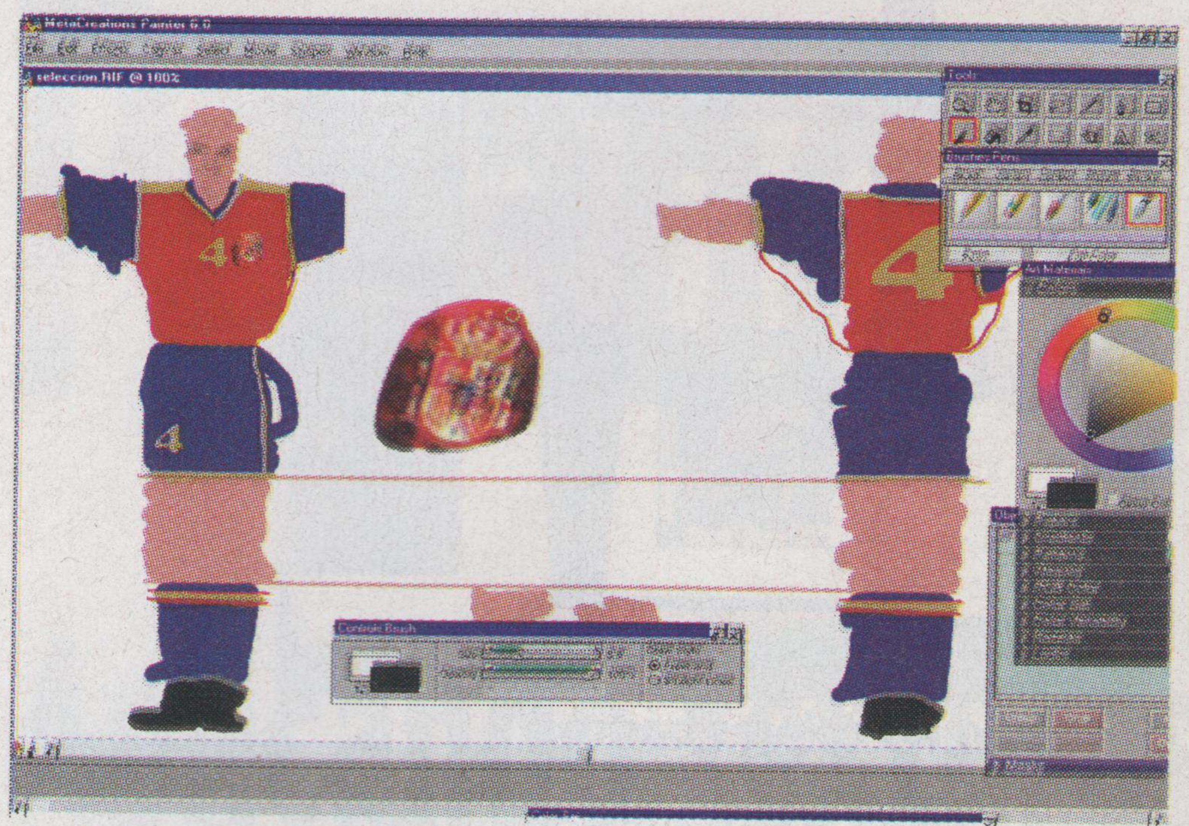
nad el color de la piel de la cara pegada (generalmente es la herramienta "cuentagotas" en todos los programas de dibujo, la que permite seleccionar un color de los que aparecen en una imagen). Ya tenéis el color de la piel de vuestro personaje tomado de la realidad. Con la herramienta de pincel, pintad las zonas correspondientes al resto de la cabeza, cuello y brazos (las piernas las dejamos para luego) tanto de frente como de espaldas. No os preocupéis por seguir los contornos de la plantilla, en la figura Poser lo único que cuentan son los polígonos que aparecen en la plantilla, así que todo lo que pintéis que sobresalga no aparece en el personaje al colocarle la textura.

Una vez que tenemos la cara, comenzamos con la camiseta. La Selección Española lleva un escudo bastante grande sobre el pecho, como dibujarlo a mano puede quitar realismo a nuestra imagen, vamos a hacer igual que con la cara del personaje: recortarlo y pegarlo. De la imagen que tengamos de jugadores de la Selección, recortamos el escudo de esa imagen y lo pegamos en nuestra plantilla. Puede suceder, al igual que ocurrió con la cara, que haya que disminuir su tamaño y rotarlo para que quede recto. El resultado final podéis verlo en la página siguiente.

> Se puede dar a la figura un toque personal añadiendoles una cara

Ahora sólo queda seleccionar el color de la camiseta nacional (de nuevo la herramienta "cuentagotas" de vuestro programa de dibujo viene en vuestra ayuda). Coged el color de la camiseta en la foto de la que hayáis recortado el escudo y comenzad a pintar nuestra plantilla Poser de rojo.

Como en la foto que he seleccionado previamente no se apreciaban todos los detalles de las mangas y pantalones, descargué otra imagen de la Selección en la que sí se veían todas las partes del uniforme. Una vez observada la foto, y siguiendo el procedimiento de tomar el color de la foto origi-



La textura completa.

nal ("cuentagotas") y utilizarlo para pintar las líneas de la camiseta de la textura, añadimos las mangas azules y los detalles amarillos del cuello y mangas. El resultado debería ser como el de abajo.

Ya podemos poner el número a la camiseta. Con la ayuda de la herramienta de texto de vuestro programa de dibujo, crear un número en la espalda y en el pecho de la textura.

Creemos pantalones cortos de la nada

Dado que la plantilla CASMAN.TIF corresponde a un personaje con camisa de manga corta y pantalones largos, debemos simular que los pantalones son de deporte. La nueva versión de Poser ya permite modificar las partes de la ropa para que incluyamos las que más se ajusten a nuestro personaje, pero con Poser 2 debemos seguir el procedimiento que os comento a continuación.

Podemos tomar una muestra del color de la piel de la cara para el cuerpo

En primer lugar. marcad el espacio que existe entre el final del pantalón corto y el comienzo de las medias del jugador. Esto es, cread dos líneas rectas horizontales que marquen el espacio que corresponde a la pierna descubierta. Dichas líneas deben extenderse desde la parte correspondiente a la figura de frente, hasta la figura de espaldas, para que después coincidan.

Posteriormente, sólo queda rellenar con el mismo color que la nueva zona que hemos creado para las piernas.

Por último, coloread del mismo modo que hemos hecho con la camiseta, el pantalón, medias y calzado del jugador. Tened en cuenta que el resultado de nuestro trabajo es el básico para crear sprites posteriores, añadiendo luces, animación y otros objetos en Poser. Tal y como hemos concluido, el trabajo quedaría estupendamente para sprites de tamaño medio, pero si queremos hacer una "intro" en formato película, con algún plano cercano, etc...deberíamos añadir nosotros mismos luces y sombras a las arrugas de la camiseta, barro a las botas, sombras a la cara y cuello, simular pelo en los brazos, etc...pero eso depende ya del sentido artístico de cada uno.

Aplicando la textura en Poser

Nuestra textura de la Selección debemos guardarla en formato BMP o bien TIF, dado que son los únicos formatos gráficos que admite Poser 2 para cargar texturas en los modelos.

Una vez la tenemos guardada con el nombre que deseemos, sólo tenemos que abrir Poser, y desde el menu "Render" seleccionar "Surface Options". Se nos abrirá una ventana en la cual podremos seleccionar en un desplegable la figura a la cual vamos a aplicar la textura ("Figure 1", etc...), y cargar las texturas correspondientes a "Bump" (ver la sección siguiente) y a "Texture Map" (la que hemos creado).

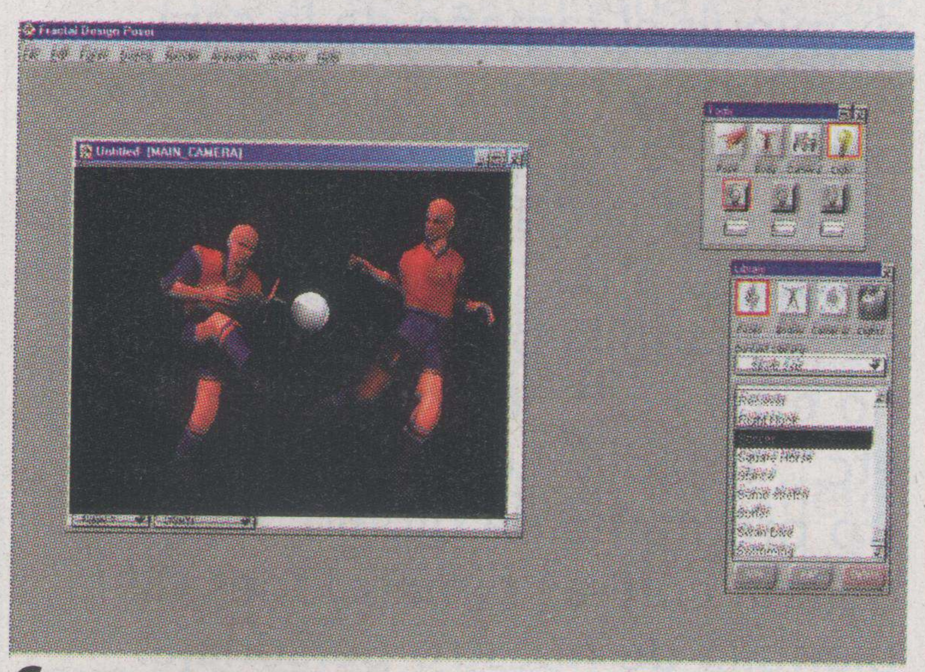
Pulsamos el botón "Load" en "Texture Map" y seleccionamos la imagen TIF o BMP de la textura que hemos guardado. A partir de ahora podremos modificar las posturas, crear animaciones, etc... con nuestro personaje, que la textura corresponderá a la que hemos indicado a Poser.

Probad a hacer un Render de vuestro personaje, y comprobaréis los resultados de vuestro trabajo previo. Con éste método podréis crear los sprites de vuestros personajes en la mitad de tiempo que en el editor de Div "a mano", con resultados infinitamente mejores.

Tipos de texturas: Map y Bump

Como comentábamos, en el cuadro de carga de texturas de Div se nos da la opción de seleccionar dos tipos de texturas diferentes. Veamos en qué consiste cada una de ellas.

Bump: La traducción literal es "abolladura", pero este tipo de mapas lo que nos indican es el relieve de la textura. No contienen color, son en blanco y negro y, según se pinte una zona más clara u oscura, nos indicará las partes más resaltadas del modelo.



Creamos una escena con nuestra textura.

Sonará lioso, pero en realidad es bastante fácil. Abrid por ejemplo, la textura "Male Muscle Texture.TIF", que corresponde con la textura Bump del hombre desnudo. Veréis una imagen en blanco y negro que simula la musculatura de un hombre.

En esta imagen, las zonas blancas son las zonas que "sobresalen" de la figura. Por ejemplo, los abdominales son casi blancos puesto que son músculos que dan relieve a la figura.

Por contra, las zonas más oscuras (gris oscuro y negro) indican "hundimientos" en la superficie de la figura. Así, la zona donde están los abdominales es más oscura que el resto, para indicar que está a un nivel inferior.

Si quisiéramos crear una textura Bump para nuestro uniforme de la Selección, construiríamos una superficie gris medio (indicaría las zonas planas), sobre las que aplicaríamos gris claro y blanco en el punto más alto de las arrugas, y gris oscuro y negro en los dobleces.

Este tipo de textura da mucho realismo a la ropa, sobre todo a los músculos de las figuras con cuerpo descubierto, pero consume tiempo de render y debemos evaluar si son necesarios para nuestra figura. Si vamos a crear sprites de 100x100 pixels es inútil realizar mapas de relieve, pues no se apreciarán. Si queremos realizar una animación con planos cortos y medios de los personajes, probablemente sea necesario crear un mapa Bump para ellos.

Los Map Textures son las texturas que hemos creado a lo largo del artículo. Definen los colores que tienen los personajes, así como los detalles de la vestimenta, etc... Una textura de color bien trabajada es la que marca la diferencia entre un personaje mediocre y uno bueno a la hora del render.

Y esto es todo por hoy. Próximamente veremos cómo diseñar la librería de animación con las posturas creadas en el capítulo anterior, y crearemos los sprites correspondientes para cargarlos en Div.

Como siempre, recordaros que podéis realizar las consultas que deseéis en el e-mail:

tayete@teleline.es.

Santiago García Mazariegos "Tayete"

El correo del lector



Para vosotros

Un número más, aquí tenéis vuestro espacio para salir a la luz pública. Para poneros en contacto con otros diveros, para intercambiar ideas o, para criticarnos o alabarnos y, en general, para decir lo que os apetezca, siempre y cuando tenga que ver con DIV claro está. No nos cansamos de repetir que esta revista es tanto vuestra como nuestra.

Crítica constructiva

Hola, quería comentar algunas cosas que me gustan y otras que no de la revista DivMania:

- El tema del concurso para enviar los juegos está muy bien, pero se limita solo a Div, somos un monton de personas que leemos la revista y programamos en otros lenguajes como C, Paskal, etc... ¿Por qué nosotros no podemos participar? Me parece una manera de limitar la revista a solo Div.
- Me gustan sobre todo las secciones que tratan sobre programacion con DirectX, Utilidades con MFCs, OpenGl, etc. De esta manera no se limita sólo a Div. En el fondo la gente que se inicia programa en Div, pero llega un momento que con Div no es suficiente y aprenden a programar en C bajo Windows. Así que estas secciones son de bastante utilidad.
- Las fotos que ponéis junto al texto me parecen demasiado exageradas, me explico: igual se está comentando cómo programar la parte de hablar con otros personajes y al lado en vez de salir una foto del ejemplo de la revista sale una foto del King Quest VIII o del Baldurs Gate . ¿¿?? Y todos sabemos que el código que se esta tratando no le llega a la suela de los zapatos de esos juegos.



Bueno, en resumen que la revista está muy bien que espero que sigáis así pero ke no os limitéis tanto a DIV o en caso contrario que penséis en crear una revista nueva *DirectXMania* o *Cmania*. Un saludo. *JVMA*

Gracias por tu carta. Tomamos nota de tus sugerencias. Respecto a la participación de juegos realizados en otros programas, sintiéndolo mucho no es posible. Sin embargo estamos abiertos a publicar cualquier juego que nos enviéis y que os haga ilusión ver publicado. Sobre la limitación a Div Games Studio, querido amigo, Divmanía está hecha por y para el Divmaniaco y aunque toquemos otros temas siempre estarán relacionados con Div. Si no lo hiciésemos así perderíamos nuestra razón de ser. Lo de hacer un DirectXManía... pues es una idea.

Dudas

Hola, soy Victor Arago, un lector de Divmanía. En el número 8 explicáis cómo hacer mapas de durezas pero no me he aclarado. Podríais ayudarme. Espero vuestra contestación.

Lo mejor es que te pongas en contacto con el autor del artículo. De cualquier modo deberías concretarnos qué es lo que no entiendes porque de otro modo no podemos ayudarte.

Mensaje de la redacción

Un mensaje de los que hacemos esta revista para Alberto Zurriaga. Para resolver tus dudas lo mejor sería que te pusieras en contacto con Hammer Technologies, los creadores de DIV Games Studio que son quienes mejor pueden informarte. Nosotros sí que podemos contestarte a dos cosas: una que lo mejor para aclararte es que te compres el DIV original (el pirateo es un delito) y que su precio es muy asequible teniendo en cuenta que cualquier software de programación se sube por encima de las 40.000 pts.

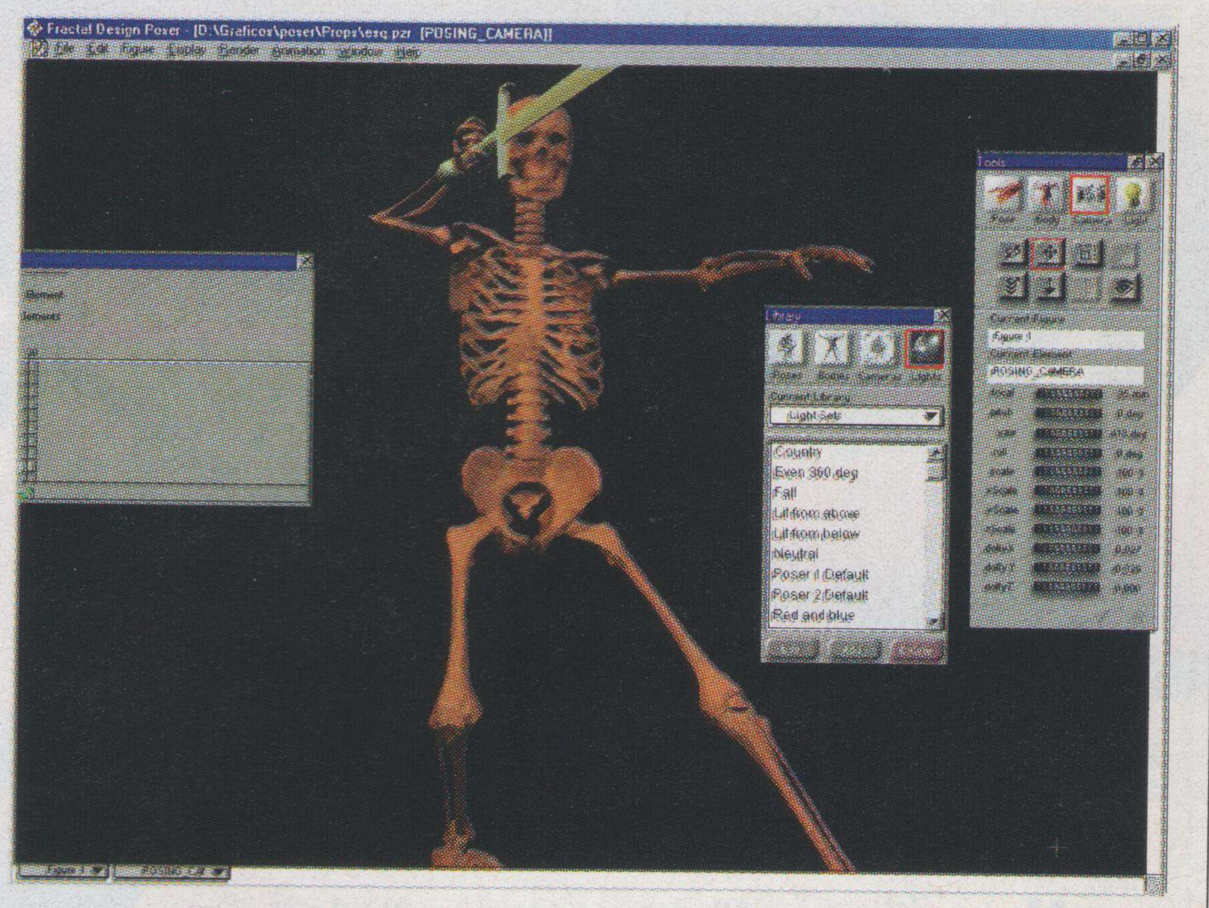
Corrector ortográfico

Hola. En primer lugar quiero felicitarles por la revista DIV manía. Para gente como yo que acabamos de emprender el viaje alucinante dentro del mundillo de la programación es una fuente inagotable de trucos e ideas.
En el CD de este mes había una corrector ortográfico. Antes de instalar la aplicación no sospechaba que iba a hacer tantas cosas. La corrección de cualquier programa en tiempo real me parecía una idea genial y la calculadora era exactamente lo que necesitaba. Sin embargo, hay un montón de funciones que no están muy bien explicadas en la ayuda. ¿Qué significa toda la información sobre la jerarquía de las ventanas? ¿Cómo puedo corregir el código fuente de C++? ¿Cómo puedo creer un diccionario nuevo con los nombres de mis amigos sin añadirlas a los diccionarios principales? *Pablo Olabarrieta.*

Está muy claro, amigo mío. Lo que necesitas es ponerte en contacto con su creador:
Anthony Rostron, ajr@arrakis.es.

Sugerencias

Hola, soy un nuevo usuario de Div2 y por casualidad vi la MAGNIFICA revista DIVmania en el kiosko, sin dudarlo un instante me la compré. Mi problema es que como no tengo los anteriores números no puedo seguir los cursos como el de la aventura grafica, RPG, etc. (menos mal que he encontrado esta página. Para mejorar esto, podrían poner en el mismo apartado de las suscripciones un mensaje(ahora no sé como lla-



marlo) para que te envíen números atrasados de DIVmanía. Gracias por vuestra atención, se despide Jordi.

Parece que todavía hay despistados. En fin, volveremos a repetir que para conseguir números atrasados debéis poneros en contacto con el departamente de suscripciones en el número 91 304 06 22 o bien en la dirección suscripciones@prensatecnica.com. Tomad nota y que no se repita.

Buscando ayuda

Lo primero que quiero hacer es transmitiros mi agradecimiento por vuestra gran labor, gracias a vosotros podemos expresar nuestras dudas, peticiones y todo lo que necesitemos. Me llamo Alberto y tengo 17 años, estoy preparando un proyecto con un amigo, llamado Juan Luis. Se trata de un juego tipo *Golden Axe*, ambientado en la Edad Media, necesitamos ayuda en lo que podáis. Necesitamos tanto grafistas 2d como programadores, si os interesa poneros en contacto conmigo en la siguiente direccion: albertoblanc@yahoo.es. No nos importa si tengas experiencia en Div, lo importante son las ganas que le echemos. Dicho esto sólo queda despedirme y daros de nuevo las gracias por brindarme esta oportunidad. Hasta pronto.

Sonido 3D

Uno de los ganadores de nuestro número pasado nos ha hecho llegar este mensaje:

He recibido un mail de un Divero que al parecer ha creado un juego en el cual hay una función llamada Dolby, que hacía un efecto estéreo o algo parecido. Me dice que mi idea del sonido 3D es demasiado parecida a la suya.

Si habéis mirado todos los archivos del CD, veréis que en uno hay un mensaje que dice algo así como que pido perdón por si alguien ha hecho una cosa parecida, por que como no tenia ningún medio por el cual enterarme, no sabía si eso ya estaba inventado.

Aun así, también me gustaría que quedase claro que el sonido 3D fue una idea totalmente mía, sin ningún tipo de plagio de ninguna parte y que mi única intención era dar a conocer una funcion que no está incluida en DIV, y que creo que es interesante para todo creador de juegos 3D. Nada mas, perdón por las molestias y gracias por todo.

Error de DIV 2

En el número pasado en esta misma sección respondíamos a un lector que nos preguntaba sobre un error de DIV al que no sabíamos responder. Pues bien, poco después, él mismo nos contó que ya había encontrado la solución que os reproducimos ya que consideramos de interés para todos:

El problema sólo existe cuando se le da a una variable el valor "con" dentro del código del programa. Hay 2 formas al menos de resolverlo:

1º. Si la variable es tipo STRING le das el valor "co" y luego le añades una "n" o le pones "conw" y borras un carácter.

2º. La otra opción es cargarlo de un archivo externo con las funciones fopen() y fread(), teniendo en cuenta que se dan las órdenes escribiendo. Por eso necesitaba la palabra "con", porque es una preposición muy útil :•).

De todas formas el DIV2 tiene más "errorcillos". Está el del sonido... que se oye MUUY bajo... y los módulos de música se oyen MUUUY altos. Además está la función roll_palette() que en DIV1 funciona pero no en DIV2. El creador de DIV nos dijo que eso se solucionaba haciendo un fade_off antes() de roll_palette() y luego un fade_on(). Claro que hay que hacerlo muy rápido para que no se note... y es mejor usar fade (100,100,99,64) para que el cambio sea lo más rápido posible. Pero no son los únicos. Hay un problema al usar unload_song() ya que no descarga entre 32 bytes y 1 kbyte de memoria del módulo. Además las funciones de red no he conseguido que me funcionen nunca ¡eh, esto me interesa, ¿sabes de alguien que sí le hayan funcionado?! *José Luis Cruz*

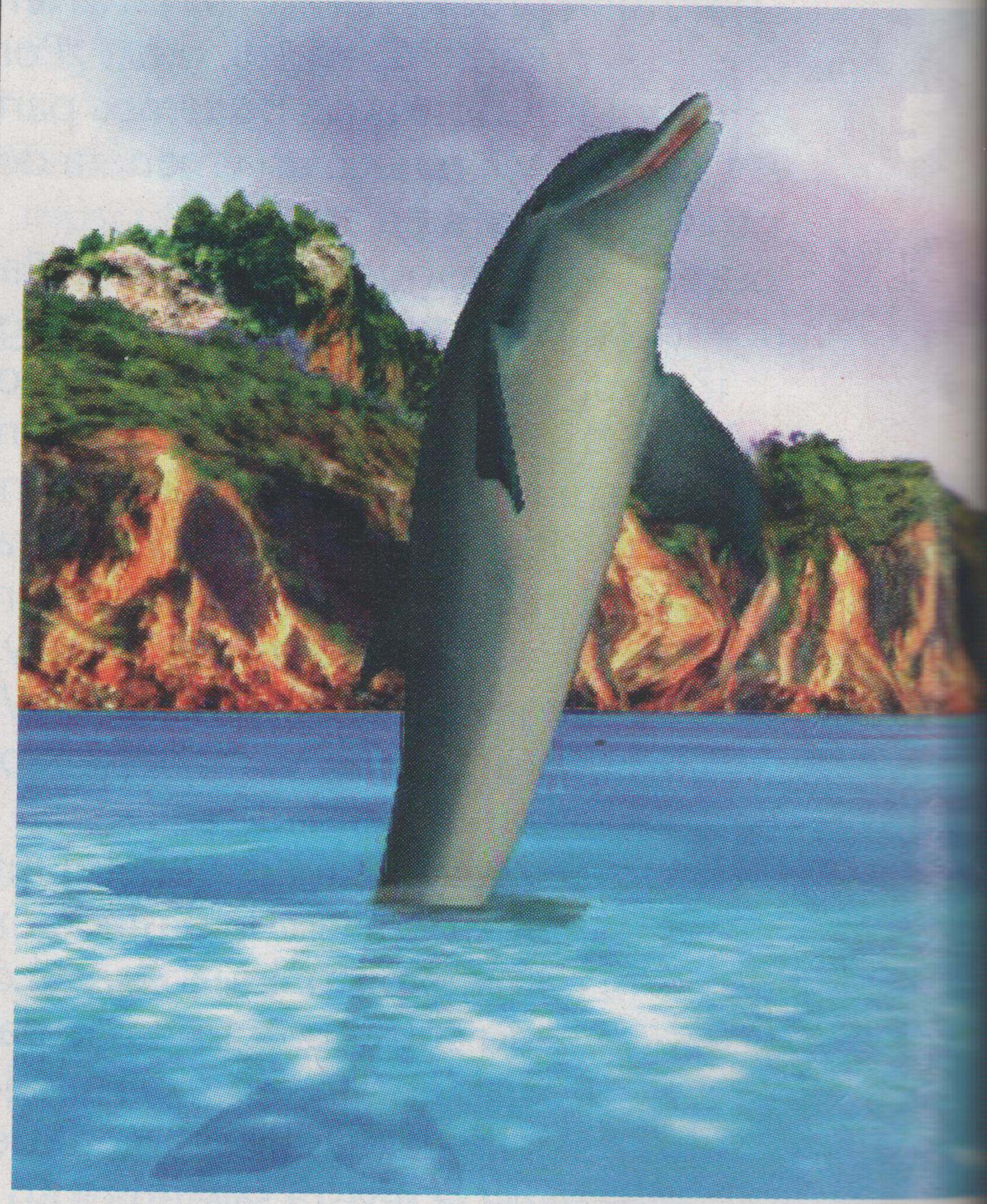
Anuncio

Busco Programadores Div. Soy de Santiago de Compostela; osea que prefiero que tú también lo seas. Yo soy músico, programador y grafista. Ocasionalmente soy escritor y artista, y miembro de la corte superior de los "idiotas ilustrados". Propongo un tipo de juego alejado del argumento estúpido –salva el mundo–, y la sugestividad por otros medios más psicológicos que el típico charco de sangre.
José A. Cerqueiro Otero. R/ Valiña nº 9 Grixoa, 15898, Santiago de Compostela, La coruña.
budabyte@de.com

Un poco de todo

Saludos, estimados amigos de Divmania, porque supongo, que nos podemos considerar todos nosotros como amigos, ¿no? Bueno, a lo que iba. En primer lugar, daros la enhorabuena por la publicación en vuestro número 6 de esos cincuenta juegos de los lectores, algo, a mi parecer, muy necesario; ya que el publicar tan solo los tres ganadores era insuficiente, o al menos a mi me lo parecía. También quiero felicitar a todos aquellos que se han ocupado de realizar los diversos cursos. *Pelegrina.*

Gracias por tus felicitaciones. Respecto a tus críticas y sugerencias, decirte que tomamos nota. Sólo una cosa, como ya hemos dicho en otra ocasión Divmanía no tiene relación directa con Hammer Technologies con lo que tus preguntas sobre el futuro de Div no podemos responderlas y respecto a las posibilidades de mejora, recordarte que ésta es una revista pequeña y, creenos, hacemos lo que podemos.



Esperamos vuestros mensajes

Para cualquier sugerencia, duda o aclaración podéis mandar vuestras cartas o correos electrónicos a la siguiente dirección:
Divmanía
C/ Alfonso Gómez 42. Nave 1-1-2
Madrid 28037. España.
Tfno: 91. 304. 06. 22
Fax: 91. 304. 17. 97
E-mail: *divmania@prensatecnica.com*

PROGRAMAR JUEGOS ES COSA DE NIÑOS SI TE SUSCRIBES

a Div Manía



Si deseas estar en la vanguardia del mundo de la informática, suscribirse a DIV MANÍA es un primer paso acertado porque...

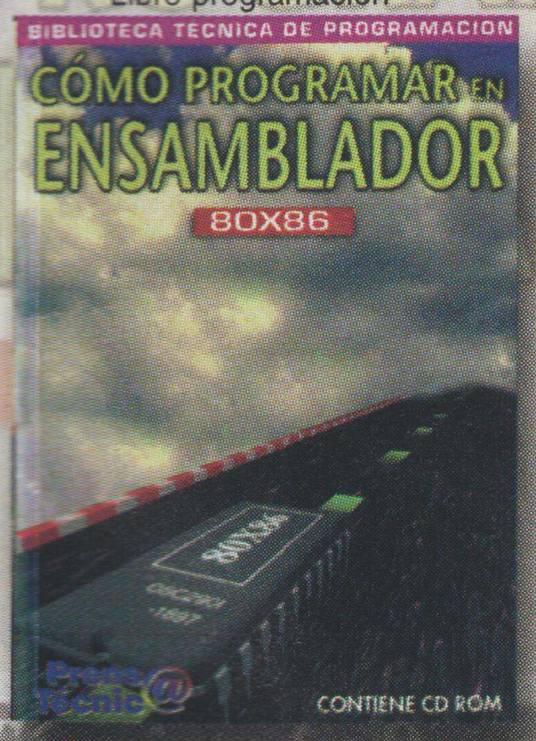
- *Es la única revista escrita por y para los programadores de videojuegos. Nuestra redacción está compuesta por veteranos desarrolladores, expertos del entorno DIV, grafistas y muchos otros profesionales del software de entretenimiento que dan lo mejor de sí a los lectores.*
- *Te ofrece lo último en el delicado campo de la programación de videojuegos, con los títulos que se encuentran en proceso y los productos recién salidos del horno o de los PCs.*
- *Para seguir avanzando hay que saber echar la vista atrás y a la vez no olvidarse del futuro, y **Div Manía** empieza un nuevo camino.*
- *Nuestra revista presenta un look muy cercano a sus lectores, salido de las inquietudes de todos vosotros.*
- *Nunca nadie te ha ofrecido tanto por tan poco; nunca has tenido tan cerca la oportunidad de estar al día de lo último en programación por el mínimo esfuerzo de acercarte al quiosco o enviar nuestro cupón y recibir la revista en casa puntualmente cada dos meses.*
- *En el interior del CD-Rom encontrarás los elementos con los que todos los programadores sueñan.*
- *Somos como tú y conocemos, más o menos, qué se esconde dentro de tu cabeza. Y si no lo conocemos aún, lo aprenderemos gracias a ti.*
- *Ofrecemos las más diversas sorpresas, las más interesantes ofertas, para que no te olvides de que la programación siempre está viva.*

Además, el suscriptor tiene derecho a la siguiente oferta:

- **Con un año de suscripción (seis números) regalamos un producto a elegir entre:**
  "Programación Gráfica para PC"
  "Cómo programar en Ensamblador"
- **Con dos años de suscripción (doce números) regalamos DIV 2**



PROGRAMACIÓN GRÁFICA PARA PC
Libro programación



CÓMO PROGRAMAR EN ENSAMBLADOR
Libro programación



DIV II
Creación de juegos

Solicite su ejemplar enviando este cupón por correo, por fax: 91 304.17.97 o llamando al teléfono 91 304.06.22 de 9:00 a 19:00 h.

CUPÓN DE SUSCRIPCIÓN ANUAL A DIV MANÍA

Deseo suscribirme a la revista **DIV MANÍA** acogiéndome a la siguiente modalidad:

☐ Suscripción: 1 año (6 números) por sólo 5.970 ptas. ☐ Correo certificado 1 año: 1.500 ptas. adicionales

☐ Suscripción: 2 año (12 números) por sólo 11.940 ptas. ☐ Correo certificado 2 años: 3.000 ptas. adicionales

Desde el número ____

Además recibiré gratis:

☐ Por 1 año de suscripción: **uno** de los siguientes productos:
☐ Por 2 años de suscripción: **DIV II**

☐ Programación Gráfica para PC ☐ Cómo programar en Ensamblador

Nombre y apellidos ....
Domicilio ....
Población ....
C.P.... ....Provincia ....
Telf. .... Profesión ....
DNI/NIF:..............................

FORMA DE PAGO:

☐ Con cargo a mi tarjeta VISA nº__________________más gastos de envío
Fecha de caducidad de la tarjeta ....
Nombre del titular, si es distinto ....

☐ Domiciliación bancaria, más gastos de envío
Población ....
Ruego a Vd. que se sirva cargar en mi:

☐ cuenta corriente
☐ libreta de ahorro número

ENTIDAD	OFICINA	DC	Nº CUENTA

el recibo que será presentado por PRENSA TÉCNICA S.L. como pago de mi suscripción a la revista **DIV MANÍA** más gastos de envío.

FIRMA:

☐ Contrarreembolso del importe más gastos de envío.
☐ Cheque a nombre de PRENSA TÉCNICA, que adjunto más gastos de envío.
☐ Giro Postal (adjunto fotocopia del resguardo) más gastos de envío.

Prens@ Técnic@ de libros y publicaciones

C/ Alfonso Gómez, nº 42 Nave 1-1-2. 28037 Madrid. Tfno: 91 304 06 22. Fax: 91 304 17 97
e-mail: maspc@prensatecnica.com http://www.prensatecnica.com

Contenido CD-Rom



En este número nueve el contenido del CD-Rom que acompaña a la revista seguro que os será de utilidad para vuestros trabajos y proyectos. Aparte de los juegos ganadores y los complementos de los artículos, como es usual, tenemos unas cuantas demos y programas que seguro que usaréis.

JUEGOS GANADORES

Este mes se han presentado algunos menos juegos a concurso, después del verano seguro que nos llegan más. De todas formas, entre los juegos que nos han llegado a la redacción había tres que eran de obligado premio aunque eso no signifique que otros no lo merecieran. Aquí tienes los títulos que se han alzado con los laureles de la victoria, disfrútalos.

Ciclis

Un simulador de ciclismo ¿original, no? Desde luego a nosotros nos lo ha parecido y mucho. Pero es que además derrocha sentido del humor (y algo de mala leche, todo hay que decirlo) y, por tanto resulta muy divertido.

El secreto de la Isla Tortuga

Un misterioso secreto que tú solo debes desvelar valiéndote de tu inteligencia. Unos novatos en el tema se lanzaron a crear una aventura gráfica y ¡mirad el resultado!

Yamis Arena

El primer juego Div que funciona en red. Una auténtica exclusiva si no fuera porque conseguir que funcione en red no es nada sencillo.

DEMOS Y PROGRAMAS

Dreamweaver 3

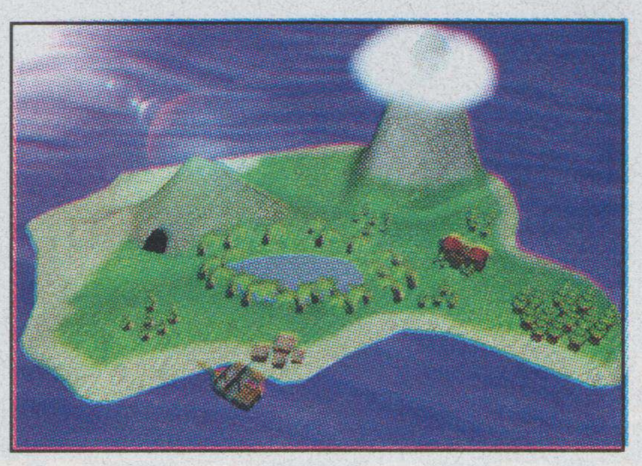
Demo de este programa profesional que sirve para crear y mantener páginas web. Su completo menú de opciones y su fácil uso te permitirán diseñar la página donde meter toda la información sobre tus juegos o temas que más te interesen. Puedes hacer páginas completas usando HTML dinámico, meter imágenes, etc., sin escribir una sola línea de código o programando las páginas como tú quieras.

Fireworks 3

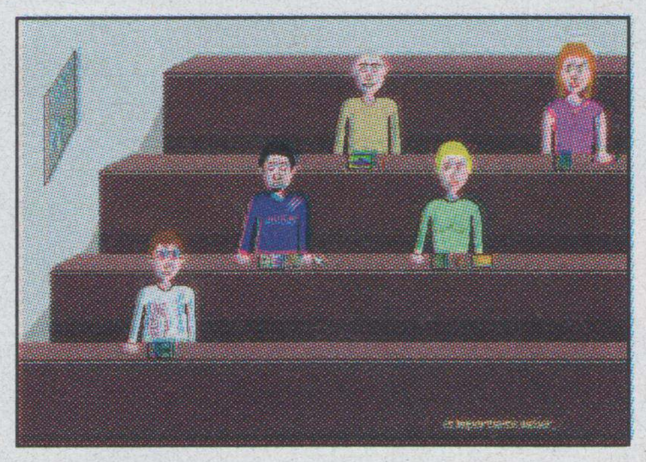
Diseña el aspecto gráfico y las imágenes que quieras meter en tu web con la demo de la última versión de este programa. Podrás encontrar múltiples herramientas de edición y construcción de todo tipo de imágenes con completos menús de ayuda que te hagan más fácil la construcción de una página web con un acabado impecable.

Dark Basic

Esta herramienta es un programa que hace más fácil usar BASIC. Dark Basic no requiere ningún conocimiento de programación, sólo hace falta imaginación para crear el software que tengas en la cabeza, desde simples editores de textos a juegos a pantalla completa y en tres dimensiones añadiendo sonidos, música, animaciones, textos y gráficos





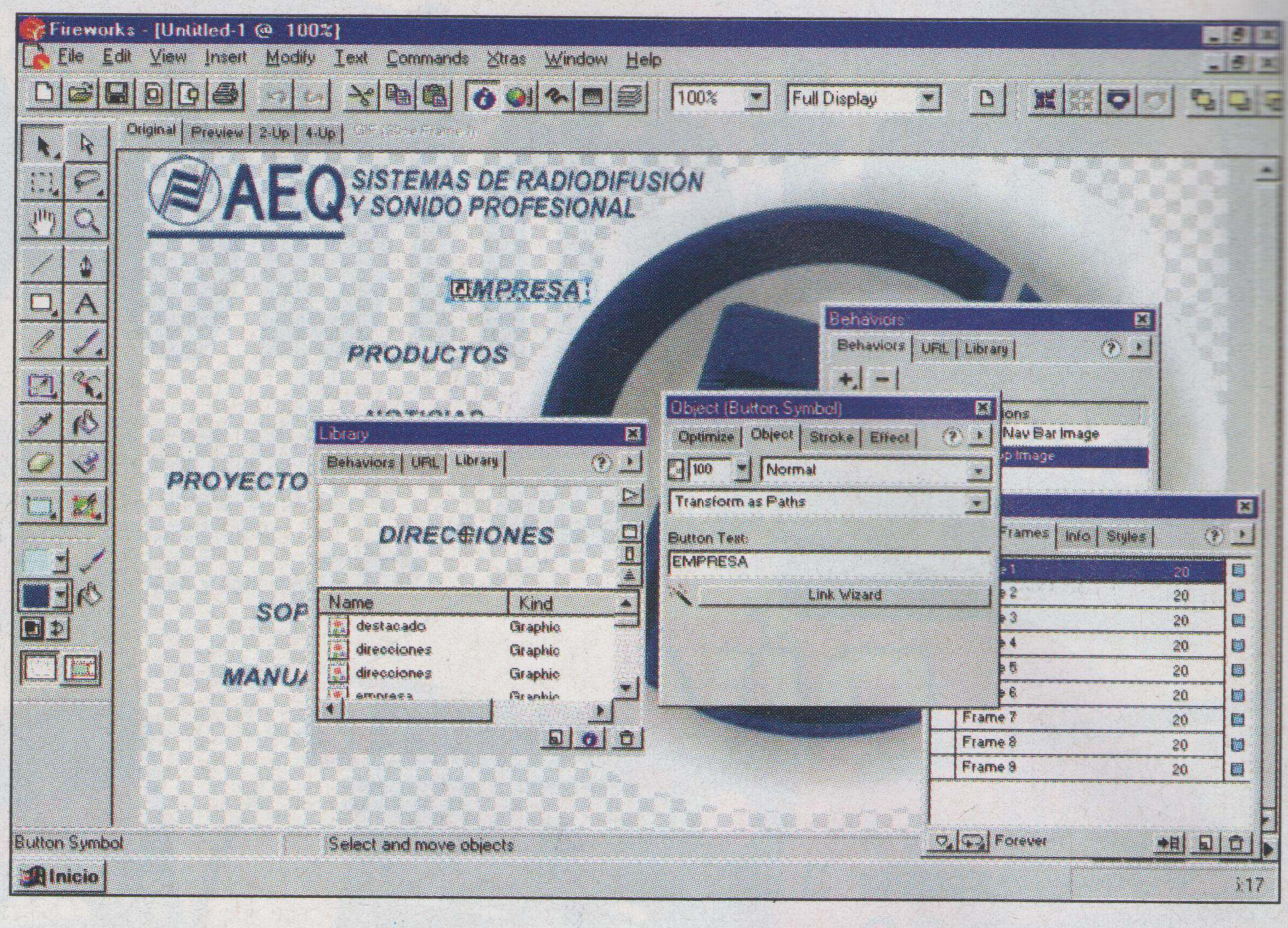




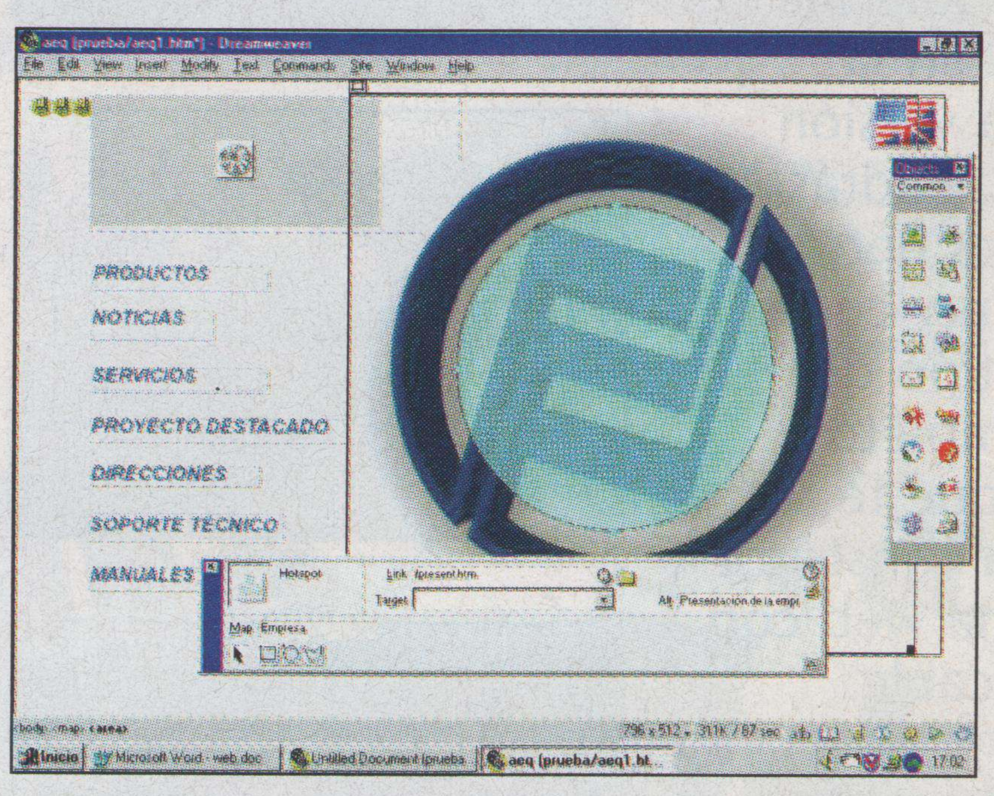


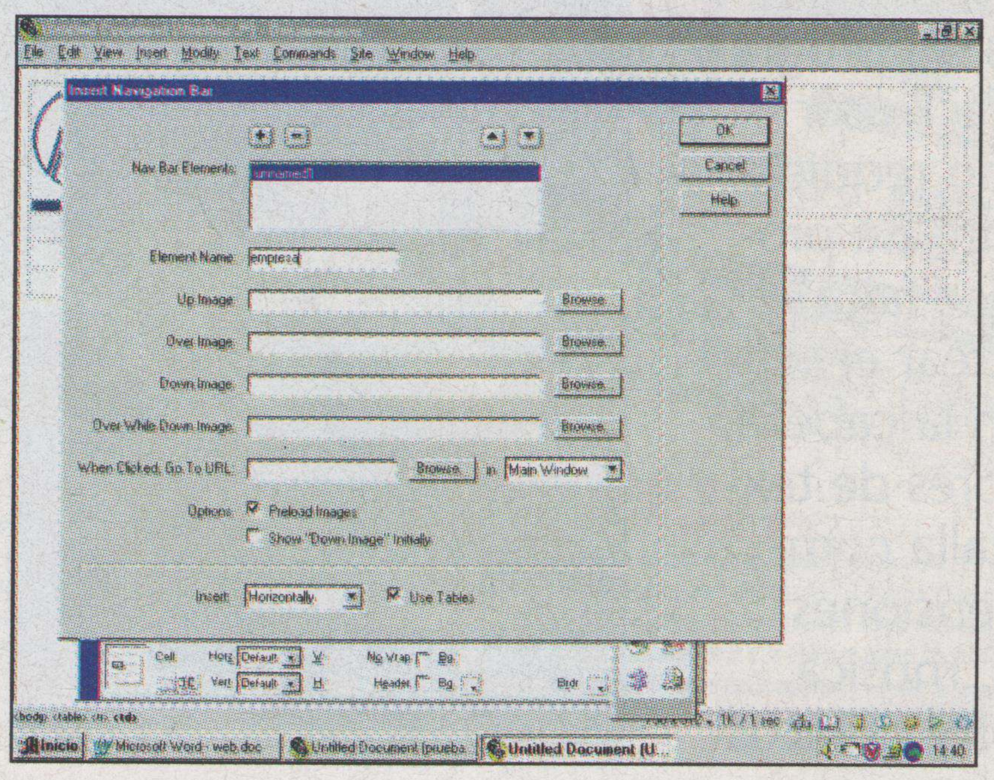






de manera sencilla. Este programa y los recursos gratuitos que vienen incluidos con él se ofrecen en exclusiva para los lectores de DIVmanía y no puede redistribuirse con fines comerciales.





M.U.G.E.N.

Con este programa podrás practicar fácilmente lo explicado en el artículo que aparece en la revista y diseñar los luchadores que quieras manejar en este beat´em up multijugador. También puedes crear los movimientos de combate que quieras que hagan y su animación respectiva. Para aficionados al noble deporte de la lucha cuerpo a cuerpo.

PROGRAMAS DE REGALO

CrannFénix

CrannFénix es un entorno de desarrollo visual basado en *Fénix*. Permite la creacción y el mantenimiento de proyectos en *DIV*, haciendo uso del compilador *Fénix* y de todas sus mejoras sobre el lenguaje original. Aporta un entorno unificado donde gestionar todos los objetos que componen un proyecto, desde los programas encadenados a los FPG´S, mapas, vídeo, etc., todo bajo entorno Windows.

Editor de textos

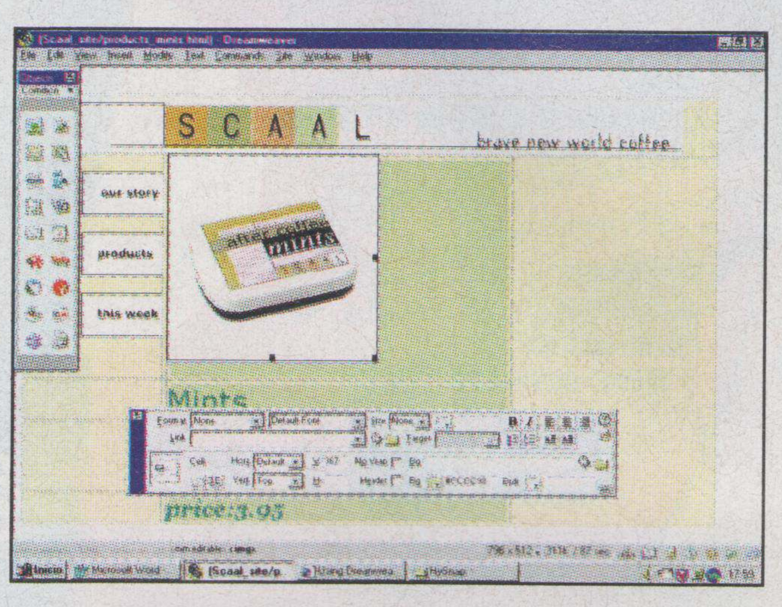
Un práctico editor de textos que nos ha ofrecido su autor para compartirlo con todos los lectores de DIVmanía.

Shareware

Este mes os ofrecemos *Palette Works* que es un programa que seguro que agradecerán los que se dedican a hacer juegos con DIV. Además podréis encontrar una completa colección del share más actual.

LIBRERÍAS

Aquí encontraréis unas cuantas librerías para que las uséis en vuestros juegos. Entre ellas, gráficos web que seguro sabéis aprovechar convenientemente en la realización de viestros nuevos proyectos y lo juegas con DIV.

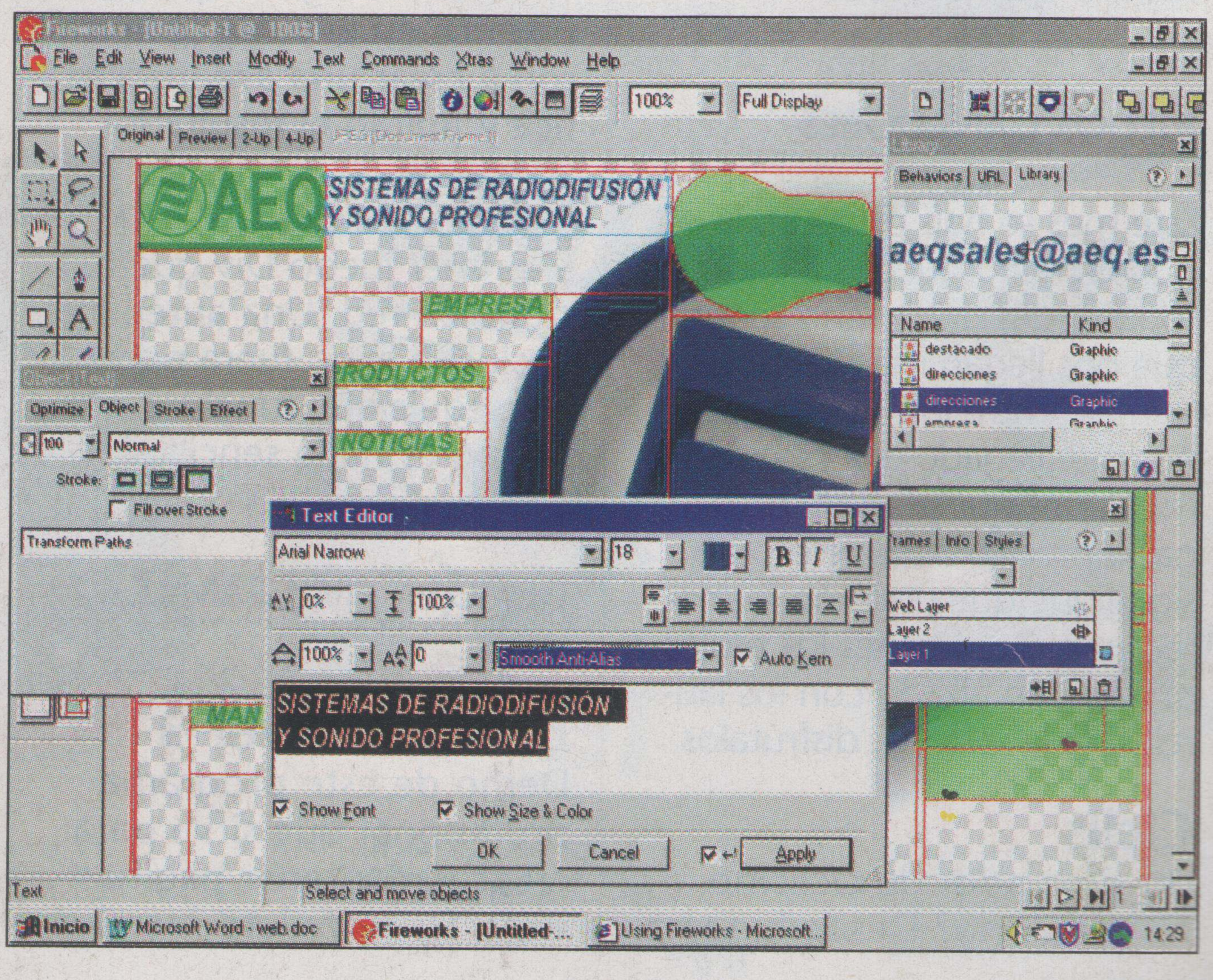


REVISTAS ELECTRÓNICAS

El número cuatro de la revista de Internet dedicada a DIV, Divnet, para todos aquellos que no puedan acceder a Internet.

CURSOS DE JUEGOS

Todos los complementos que necesitas para seguir todos las lecciones de nuestros cursos. Casi todas las secciones llevan códigos que por su extensión no es posible meter en la revista. En el Cd-Rom los encontrarás enteros y podrás seguir las explicaciones comodamente.





Space Bunnies
La nueva heroína de los juegos 3D
MUST DIE
Sólo
18,00 €
PC CD rom
2.995
Ptas
De venta en quioscos, grandes superficies y tiendas especializadas
Manual en castellano
PC CD rom
COMPATIBLE WINDOWS 98
ripcord GAMES
Digital Dreams MULTIMEDIA
Para más información, llamar al teléfono +34 91 304 06 22
Digital Dreams Multimedia
C/ Alfonso Gómez, 42, nave 1-1-2
28037 Madrid (España)
Fax: +34 91 304 17 97
www.spacebunnies.com

CONTENIDO DEL CD ROM

HERRAMIENTAS PARA TUS JUEGOS



DEMOS

Dreamweaver 3
Demo de este conocido programa de creación de páginas Web.

Fireworks 3
Diseña el aspecto gráfico y las imágenes que quieras meter en tu web con la demo de la última versión de este programa.

Dark Basic
Esta herramienta hace más fácil usar BASIC. Dark Basic no requiere ningún conocimiento de programación, sólo hace falta imaginación para crear el software que tengas en la cabeza.

M.U.G.E.N.
Un práctico programa para los amantes de los juegos de lucha.

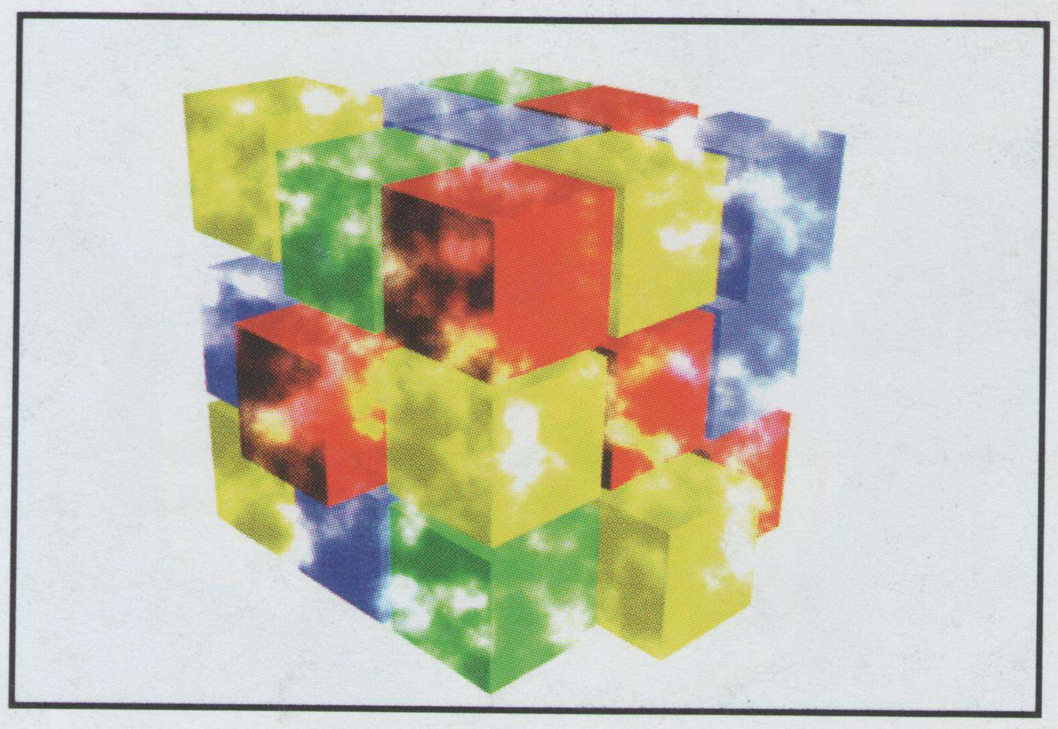
JUEGOS GANADORES
Este mes son Ciclis, El secreto de la Isla Tortuga y Yamis Arena. Muchas felicidades a todos.

CRANN FÉNIX
CrannFénix es un entorno de desarrollo visual basado en *Fénix*.

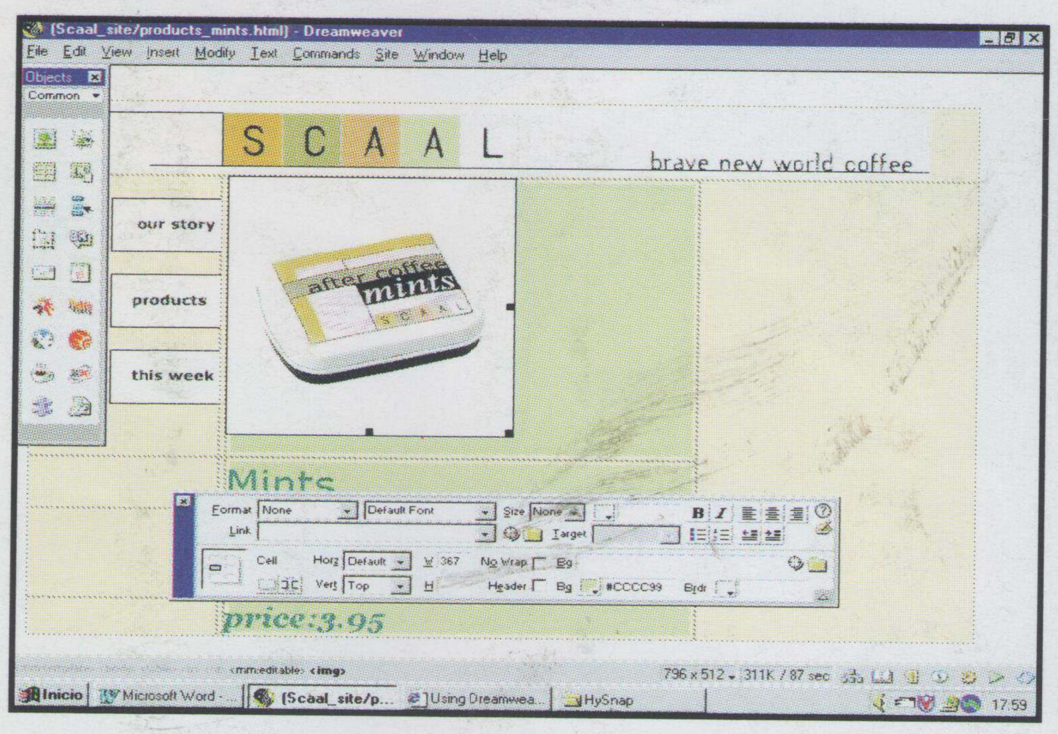
SHAREWARE
Este mes os ofrecemos Palette Works , un programa que vosotros nos pedisteis, además de una completa colección del share más actual.

Revistas electrónicas
El número cuatro de la revista de Internet dedicada a DIV, Divnet, para todos aquellos que no puedan acceder a Internet.

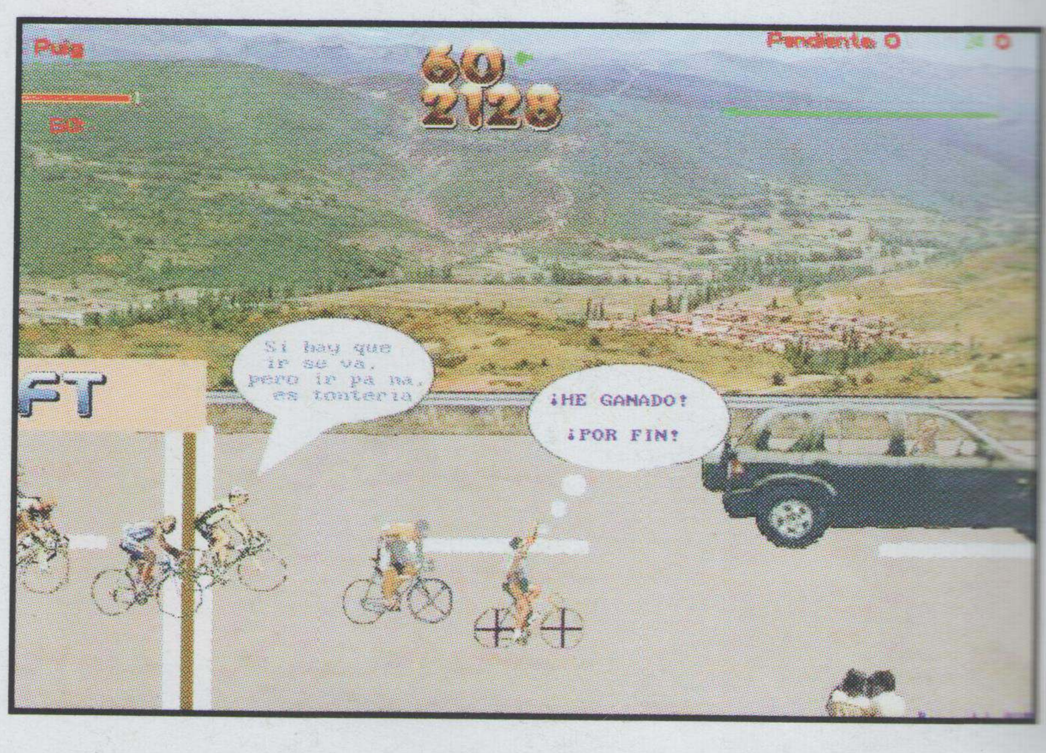
REPORTAJES. Todo sobre Stratos, una asociación de desarrolladores que a buen seguro conoceréis.



PROGRAMAS. Herramientas para el diseño de páginas Web.

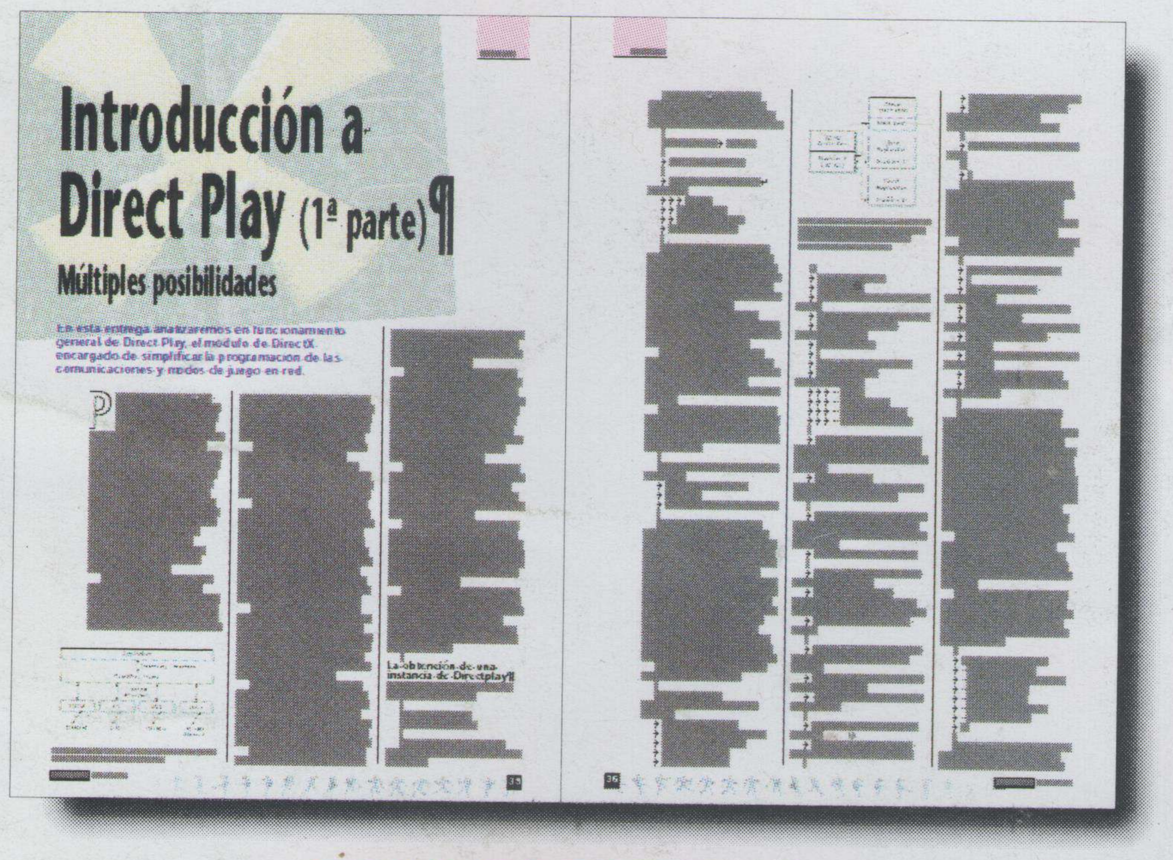


CONCURSO DE JUEGOS. Los afortunados juegos que se han alzado con la victoria.



DIV manía CON EL MEJOR CONTENIDO

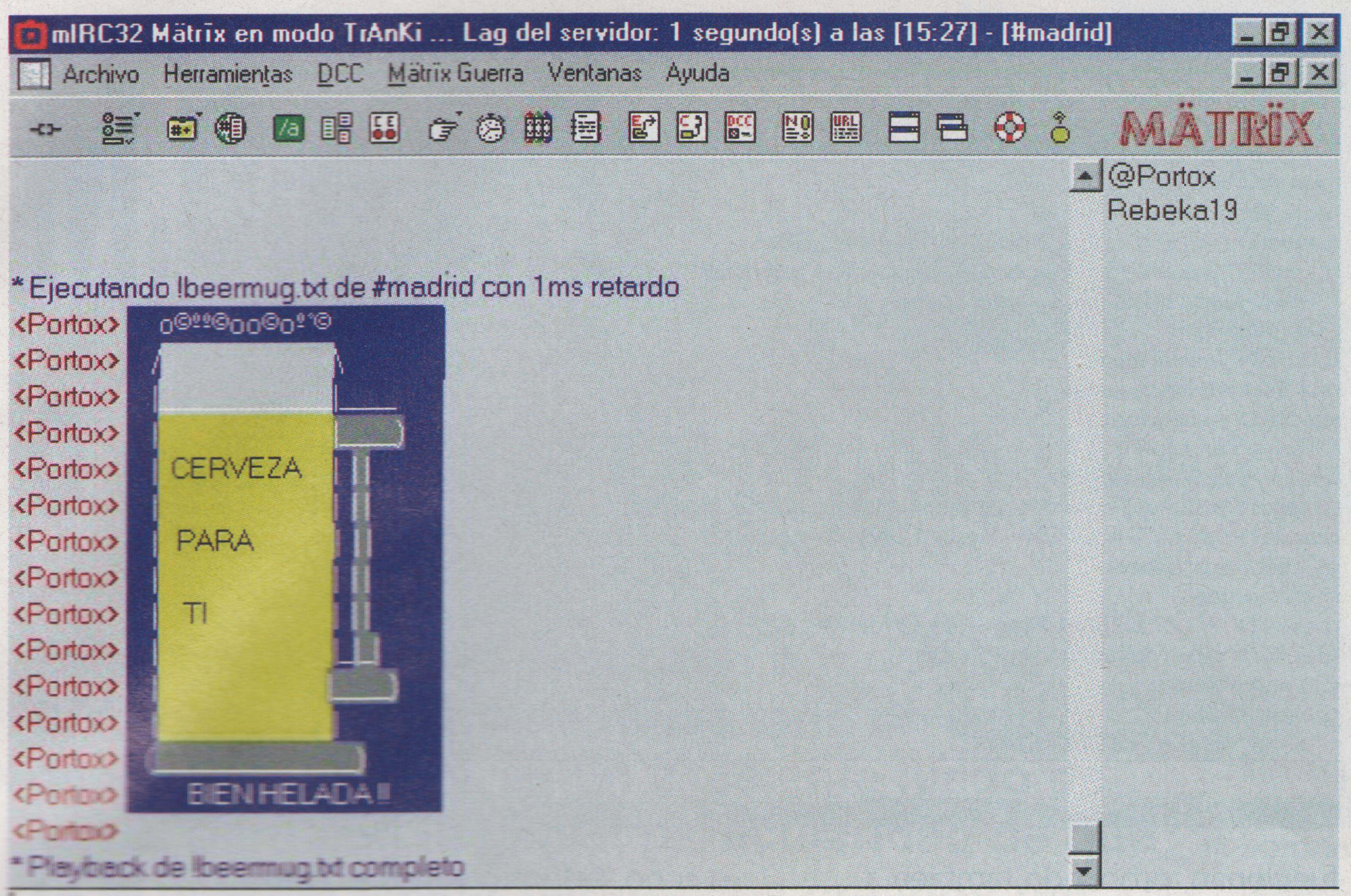






ACTUAL EXHAUSTIVO DIDÁCTICO

Y MUCHO MÁS...



...¶¶¶¶¶¶¶¶
Código 667 CODE: \´ç%¡_´¡
Código Total Eclipse CODE:
ôª__@ñª@ - ITExENI

Como podéis observar, para cualquier persona que no posea el código de desencriptación le sería imposible conocer la palabra que hemos escrito ya que sólo aparecen signos sin sentido. Este método es muy útil para comunicarnos a través del IRC con otra persona sin temor a que alguien pueda estar leyendo lo que escribimos.

> Hay posibilidad de codificar nuestra conversación con una persona para que otros usuarios no la lean

- ALERTA DE NICK: Algunos scripts incluyen esta interesante opción que tiene la finalidad de saber si alguien está hablando de nosotros o a nosotros cuando no nos encontramos presentes. Cuando nos encontramos en varios canales es muy difícil poder atender a todos a la vez y hay veces que si nos dicen algo no nos enteramos por que estamos en otro canal chateando, con esta opción, en cuanto alguien escriba nuestro nick en un canal, ya sea para llamarnos, para comunicarnos algo o, aprovechando que no nos encontramos presentes, para meterse con nosotros, el script nos avisará automáticamente.

- SISTEMA AWAY: Hay veces que por determinadas razones no podemos atender a un canal, bien por que estamos muy ocupados chateando en otro, porque estamos navegando por Internet, por que estamos muy ocupados con un privado o simplemente porque tenemos que ausentarnos un momento del ordenador. Este sistema tiene la función de un contestador automático, cuando una persona se pone AWAY, es decir, activa el sistema AWAY, automáticamente si se desea su nick cambia y pasa a aparecer en él la palabra AWAY al final, para advertir a cualquier usuario que no se encuentra presente. Aparte de eso se puede hacer que cada cierto tiempo el script haga aparecer una frase en la que se comunique al canal que nos encontramos ausentes y la razón por la que nos encontramos ausentes. Aún así, si alguien nos escribe un mensaje, queda grabado para que cuando volvamos podamos leerlo y además, a la persona que nos envió el mensaje, automáticamente se le devuelve otro advirtiéndole que nos encontramos ausentes pero que su mensaje ha quedado grabado y que se le contestará en cuanto sea leído.

¿Qué es un OP?

Es el usuario encargado de velar por el orden y mantenimiento del canal. Se identifican por la arroba "@" que tienen junto a su nombre y tiene el poder para echar a cualquier persona del canal, lo que se conoce como kick o kickear, o para echarla sin permitir que pueda volver a entrar, lo que se conoce como ban o banear. También pueden hacer OP a otros usuarios del canal.

Algunos scripts

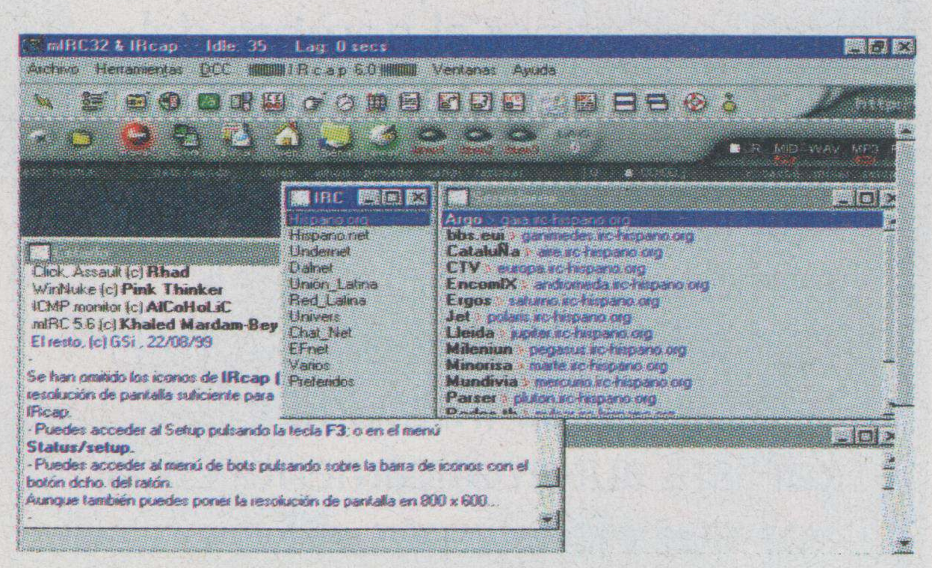
Ya hemos hablado de casi todas las características que incluyen los Scripts, pero a la hora de elegir qué Script usar cada usuario tiene que elegir el que mejor se adapte a sus necesidades, un Script muy utilizado y famoso es el IRCap debido a su interfaz gráfica y a que posee un menú muy cuidado desde el que podemos configurar todo el Script, otro también muy famoso y utilizado es el Orbital Script que incluye más opciones que el primero pero no es tan fácil de configurar y usar.

Estos son los Scripts más utilizados, de habla hispana claro, por que no debemos olvidar que existen también muchos Scripts extranjeros y no por ello menos importantes, como por ejemplo 7Th Sphere que es el Script extranjero usado por excelencia y además es muy bueno. Si deseáis buscar un Script a vuestra medida os aconsejo que visitéis http://www.guitarra.net/irc/scripts.htm. En esta pagina se encuentran la mayoría de los Scripts existentes ordenados alfabéticamente, además se incluye una pequeña descripción de cada Script para saber si nos interesa o no, en esta pagina podéis encontrar Scripts desde para los más novatos hasta para los más expertos.

Parches y accesorios

Los parches son pequeños programas para mIRC que lo que hacen es añadir nuevas propiedades al mIRC o al Script, son muy útiles para añadir nuevas e interesantes opciones al Script que estemos utilizando, así tenemos parches que aumentan la velocidad en el envío de archivos, que nos alertan cuando una palabra que hayamos configurado previamente sea nombrada en el canal e incluso los hay que son capaces de mantener una conversación con otros usuarios del IRC sin que nosotros tengamos que intervenir para nada (muy útil si alguien no para de hablarnos y no tenemos ganas de hablar con él).

Los accesorios en cambio son programas individuales que, aunque funcionan sin estar usando el Script, aportan nuevas propiedades a éste, algunos accesorios pueden



ser programas para escribir con colores, Mail-Bombings, armas de ataque, programas para conectarse anónimamente, etc.

Una buena pagina cargada de Parches y Accesorios es http://www.guitarra.net/irc/parches.htm Desde aquí podréis bajaros los últimos parches y accesorios para el mIRC.

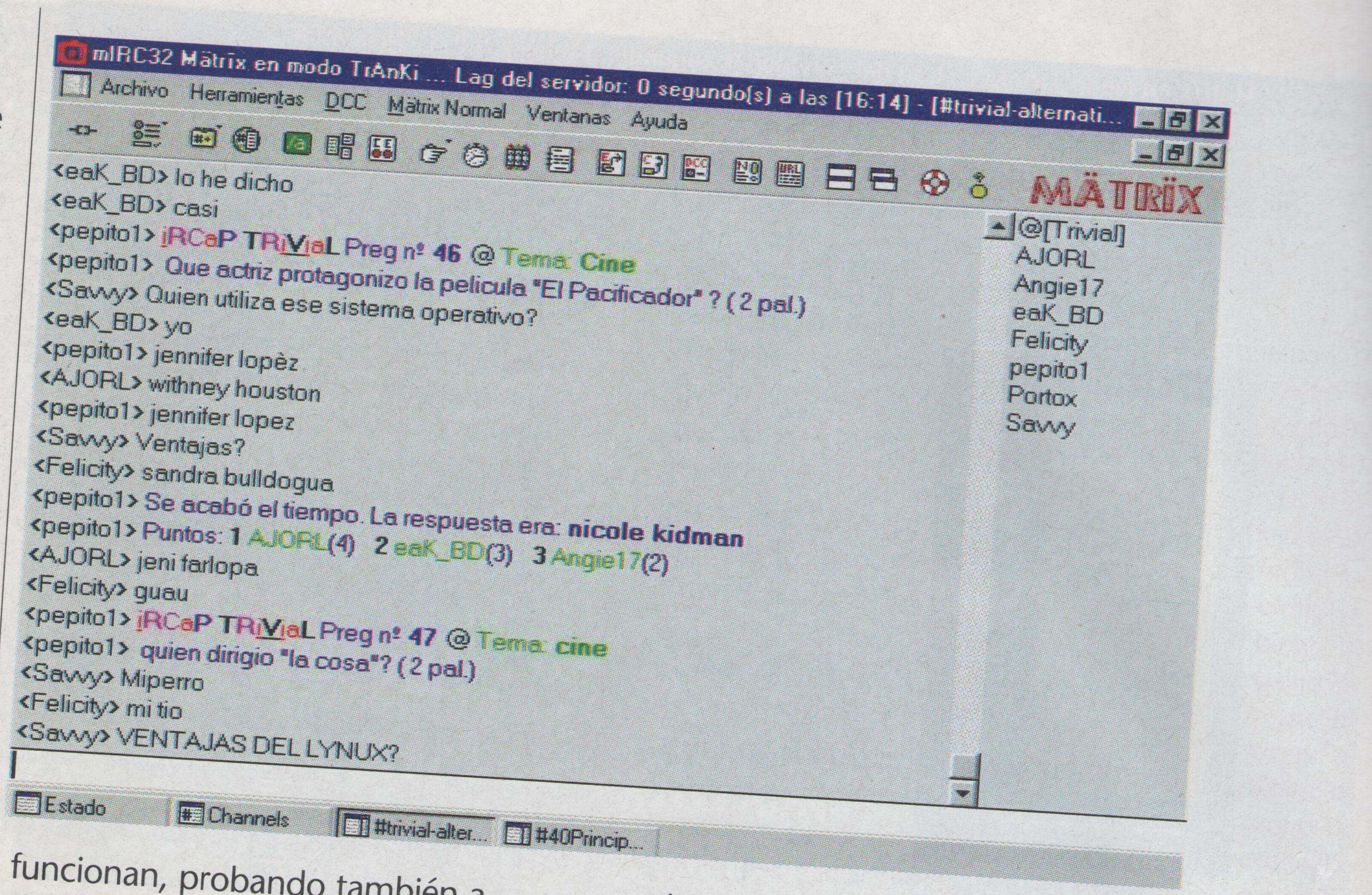
Scriptting

Se conoce con este nombre a la técnica de programar o crear Scripts, así como el lenguaje utilizado para crearlos. Aprender a crear tu propio Script no es una tarea difícil si tenemos en cuenta que el lenguaje utilizado es muy simple y que sólo nos hace falta un editor de textos. Simplemente hay que saber que existen tres menús que tenemos que crear, los Alias, los Popups y los Remotos. También tenemos que crear las variables que utilizaremos en nuestro programa.

El tema de Scriptting es demasiado largo debido al innumerable número de sentencias y combinaciones, por lo que no podemos desarrollar el tema mucho mas, pero si os interesa existe una pagina muy buena que explica muy bien paso a paso como crear tu propio Script la dirección es: http://www.guitarra.net/irc/scriptting.htm y si tenéis alguna duda podéis escribirme y estaré encantado de ayudaros. Si aún así no sabéis bien cómo empezar, lo mejor es bajarse un Script ya hecho para estudiarlo y comprender mejor cómo

¿Cómo instalar un parche?

Los parches no son como los accesorios que son programas que se pueden ejecutar directamente, los parches son archivos con extensión *.mrc que para que se instalen en el mIRC o el Script han de ser cargados previamente, para ello debéis copiar el parche al directorio donde tengáis instalado el Script, luego ejecutar el Script y escribir: /load -rs parche.mrc entendiéndose que en parche.mrc deberéis poner el nombre del parche a instalar. Así, si por ejemplo fuéramos a instalar un parche llamado speedup.mrc escribiríamos /load -rs speedup.mrc una vez hecho esto el parche queda instalado y se cargará automáticamente cada vez que ejecutemos el Script.



funcionan, probando también a modificarlo a nuestro gusto, de hecho, la mayoría de los Scripts están basados en otros previos.

Razones por las que usar un Script

Ya hemos hablado de la cantidad de opciones que puede incluir y tener un Script, pero también hay que advertir que, para un usuario poco avanzado en el campo de la informática o en Internet, puede costarle empezar a manejar un Script, aunque sin lugar a dudas os animamos a que los uséis debido a que aporta mucha más comodidad a la hora de chatear y sobretodo mucha más rapidez que hacerlo desde una pagina Web.

Un punto que puede resultar negativo a la hora de usar un Script es que no existen muchos servidores de habla hispana por lo que tendremos que añadirlos nosotros. Pero eso no es ningún problema acontinuación os escribimos algunos de los servidores mas conocidos e importantes de habla hispana y sobre todo de España junto con el puerto de conexión:

irc.cadena40.es 6667 | Chat5.telecinco.es 6667 | irc.catalunya.net 6666
pegasus.irc-hispano.org 6667 | es.unionlatina.org 6667 | irc.arrakis.es 6664 | interlink.es.unionlatina.org 6667 | snemeis.univers.org 6667 |
irc.milenium.com 6667 | namek.univers.org 6667 | irc.jet.es 6667 | irc.irc-hispano.net 6667 | irc.irc-hispano.org 6667 | irc.satlink.com 6667

Para añadir estos nuevos servidores al Script tenéis que pulsar Alt+O una vez ejecutado el Script, o si no, seleccionar en el menú archivo el submenú de opciones, seleccionar la categoría Conectar y pulsar en el botón Añadir, una vez hecho esto, en la casilla Descripción escribir el nombre que queráis para reconocer el servidor, en Servidor IRC escribir el nombre del servidor que queráis añadir, en Puerto escribir el número de puerto que aparece a la derecha del servidor que habéis añadido y después pulsar el botón añadir. Ya hemos añadido el nuevo servidor ahora bastará seleccionarlo y pulsar en el botón Conectar Servidor IRC para conectarnos a él.

Si tenéis alguna pregunta o duda de cómo usar un Script o queréis aprender a crearlos, sólo tenéis que escribirme un e-mail y yo estaré encantado de ayudaros.

Daniel García Alonso (PORTOX)
Dagal@Eresmas.com

¿Qué es exactamente un Servidor?

Un servidor es el lugar donde nos podemos conectar para chatear, cada servidor esta formado por un número determinado de canales y cada usuario que se conecta a él tiene la posibilidad de entrar a cualquiera de estos canales o de crear otros nuevos. Para hacernos una idea rápida, digamos que cada servidor es como un edificio y cada canal de ese servidor es como cada habitación de ese edificio de manera que cada cual se puede meter en la habitación o canal que más le interese de ese edificio o servidor.



M.U.G.E.N.

O cómo hacer tu propio juego de lucha

Este programa está destinado a todos los DIVeros que seáis auténticos fans de este género. M.U.G.E.N. es un excelente juego de lucha, pero con la genial particularidad de que se le pueden añadir personajes creados por uno mismo o descargados de Internet.

De esta manera se ha convertido en el juego con más luchadores de la historia, y en la red podéis encontrar personajes de todas las sagas, desde Ryu o Athena a Megamán.

Este *engine* de juegos de lucha programado por el grupo Elecbyte nos permite crear personajes a nuestro gusto, sin otra limitación que nuestra imaginación y capacidad técnica. Podremos crear desde luchadores sencillos a los más complejos con ataques especiales, *supers*, proyectiles, *fatalities* e incluso con otros luchadores o animales como ayudantes. Asimismo podremos crear escenarios con su música de fondo y añadirlos al programa. Pero lo más atractivo de todo es la posibilidad de enfrentar nuestros luchadores a otros creados por otras personas, en un plantel inmenso que incluye a los luchadores más famosos de la historia de los videojuegos.

De hecho, la modalidad más popular es la de convertir personajes de otros juegos de lucha a M.U.G.E.N. Los juegos más usuales para esta conversión son los *King of Fighters* y los *Street Fighter*, y en Internet podemos encontrar a casi todos los luchadores de estos juegos. Instalarlos para poder jugar con ellos es tan sencillo como descomprimirlos en un directorio propio de *mugen\chars* y añadirlos a la lista de personajes que hay en *mugen\data\select.cfg*.
Aún así, debo advertiros que crear un luchador para M.U.G.E.N., así como convertirlo de otros juegos, es una tarea ardua, que os llevará varios días y os producirá algún que otro dolor de cabeza. El programa está aún en fase beta (aunque es plenamente funcional), y entre otras cosas la documentación sobre cómo crear luchadores es bastante escasa. Pero en este artículo hallaréis lo fundamental para iniciaros.

Para empezar hay que decir que para jugar sólo necesitáis el programa principal, pero para crear luchadores necesitaréis más cosas. Son imprescindibles las *M.U.G.E.N. Tools*, creadas también por Elecbyte. Asimismo es más que recomendable que para empezar utilicéis el luchador básico, que encontraréis en *player.zip* y que contiene los archivos mínimos, así como explicaciones sobre su funcionamiento. También resulta muy útil un programa llamado *Mugen Character Maker*, programado por un español, que simplifica bastante la creación de personajes. Todo esto lo encontraréis en el CD de DIVManía.

Otra cosa que necesitáis (aparte de paciencia) es un bloc de notas para apuntar la larga lista de *sprites*, animaciones y movimientos del personaje. Si no lo hacéis será mucho más difícil cambiar cosas después.

Los archivos necesarios

Como ya he dicho, los archivos que definen cada luchador están en un subdirectorio de *chars*. Hay varios archivos pero básicamente se reducen a tres clases: los ficheros de gráficos y sonido, que se crean con dos programas auxiliares; los archivos con las paletas, es decir, los colores de los luchadores; y finalmente, toda una serie de ficheros de texto que definen el comportamiento del luchador, desde las animaciones que puede hacer en cada momento a los botones que deben pulsarse para realizar un ataque especial. Los veremos ahora uno por uno.

Supongamos que nuestro luchador se llama *luchador*. Así pues, deberemos crear un directorio *mugen\chars\luchador* y en él deben estar los siguientes archivos:

-*luchador.def*: Este es el primer archivo que necesita el programa, pues contiene la información básica: el nombre del luchador y los archivos que lo componen. Lo podéis crear en un editor de texto cualquiera, modificando el archivo *player.def*.

-*luchador.sff*: Es el fichero de gráficos, que contiene todos los *sprites* del personaje (algo así como los *.fpg* del DIV). Para crearlo deberemos utilizar el programa Sprmaker que viene con las *Tools*. Este programa creará el *.sff* a partir de ficheros *.pcx*, que podréis crear con cualquier editor gráfico. Deberemos organizar los *sprites* por grupos, para que sea más fácil utilizarlos después. Además deberemos introducir las coordenadas del punto base del luchador, que está centrado y en el suelo (ver Fig. 1).

Para que introducir todos esos datos no se haga pesado, se puede automatizar el proceso con

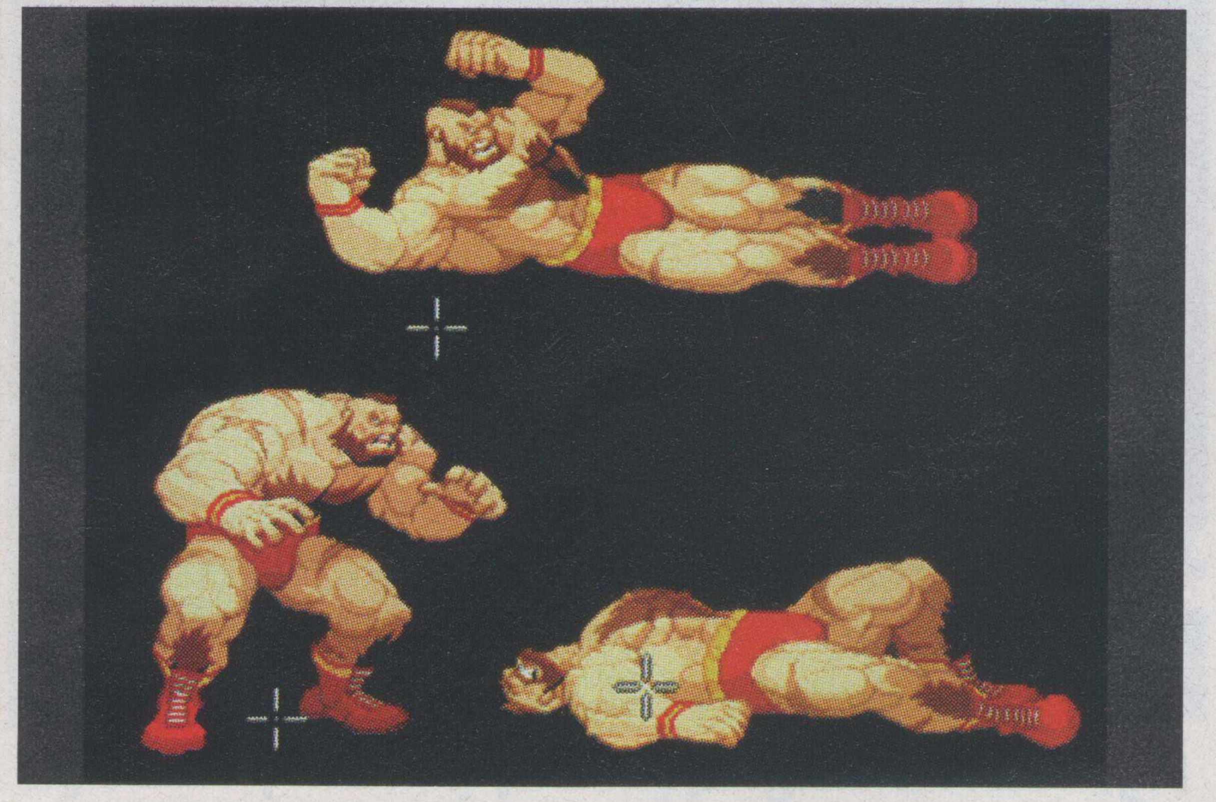


Fig. 1: Posiciones correctas del punto base.

todas y cada una de ellas imprescindibles para el correcto funcionamiento y movimiento del personaje por el mundo 3D. Ahora, de forma más detallada, os explicaremos cada una de ellas:

La preparación del proceso: consiste básicamente en la declaración de las variables privadas y la asignación de los valores iniciales del jugador para el modo8. Por ejemplo, las constantes de m8.height, m8_step, radius, etc... en cuanto a los valores iniciales, podemos encontrar "como=1", variable que en ese valor nos indica que el jugador está "vivo"; también el teletransporte a una flag de inicio al azar (con su correspondiente efecto de teletransporte, basado en procesos "dot") y la inicialización de variables como la vida (a 100), la munición, el arma escogida por defecto...Os aconsejo que le echéis un vistazo a los comentarios que hay en la declaración de variables privadas de jugador(), ya que aclaran bastante su utilidad.

Muestra de la munición, vida y blindaje: son unos simples writes, hay que considerar que el que muestra la munición tiene que hacer un write_int (y necesita un dato que no sea tabla, de ahí la existencia de mun_vab), y tiene que conocer qué arma hay seleccionada y si no se está cambiando de arma.

Restauración de pantalla: seis líneas banales que no permiten que se quede la pantalla en el fade cuando se cogen objetos, armas, etc.

Gestión del Quake Damage: perfectamente explicado en el listado; esta parte controla la variable COUNTER (contador de Quake Damage) y se encarga de terminar el Quake Damage al alcanzar el límite, para ello muestra un mensaje.

Gestión de la gravedad: una parte algo larga y quizá complicada. Toma las alturas del techo y el suelo y las divide en dos partes. Una de ellas, si se toca suelo (la variable Z es igual a la altura del suelo) o no, y la otra, si se toca techo. En la primera se controla el "balanceo de visión" (que se produce al saltar desde muy alto; la vista se va desplazando hacia abajo) y se resetea la gravedad a 0 o, en caso de que no se toque suelo, se aumenta el balanceo y la gravedad. En la segunda, si se toca techo (obviamente sólo se puede hacer tras un salto) la gravedad se restablece a 1. ¿Por qué? Por si todavía quedara gravedad negativa (vamos, que sube el proceso), para que no se quede "pegado" al techo un rato, sino que nada más tocar el techo comience a bajar.

Control del salto: es muy simple. Si se pulsa el botón derecho del ratón y se está tocando el suelo (porque si no podríamos volar), la gravedad adopta un valor negativo (-25) y el balanceo de visión se restablece (100).

Control de pulsación de teclas: aquí hay que explicar cómo funciona "armaa". Armaa es una variable que indica el arma seleccionada y su estado. Así, el valor 10 corresponde al láser de impulsos en estado "normal", el 11 a "sacando el láser de impulsos", 12 sería "recargando el láser de impulsos", 13 "esperando tras disparo de láser de impulsos", 14 "disparando el láser de impulsos". Con la Qk22 sería con la decena del 20, la ADC la del 30, etc. No todas las armas usan todos estos valores, de todos modos.

En esta parte, lo que se hace es comprobar si se pulsa una tecla de las de selección de armas (1-7) Y si "armaa" termina en 0 o en 1 (arma lista o cambiando de arma). Para hacer esta comprobación se usa el MOD (resto). IF ((Armaa MOD 10)<2) significa eso mismo, si el resto de la división "armaa entre 10" es menor que 2... Si se cumple la condición, se puede cambiar de arma, y por ejemplo si se pulsa 3 (ADC) "armaa" pasa a valer 31. Otra parte del proceso se encargará de guardar el arma actual y sacar la ADC.

Control de dirección: es algo largo pero básico y fácil de entender. El proceso del jugador avanza, luego comprueba si ha colisionado con algún bot o alguna caja explosiva mediante los get_id y si es así, retrocede lo avanzado. Funciona igual para el avance hacia delante, atrás, y el "strafe".

Control de colisiones con objetos y armas: es realmente la parte más larga pero también es muy fácil de entender. Si se está tocando suelo (único modo del que se puede recoger un objeto/arma) se van comprobando las colisiones con los distintos objetos y armas, uno por uno, y si se choca con alguno se "coge" (añadiendo el arma, la munición o lo que sea), se hace un fade y se muestra el mensaje de "recogida", diciéndole al proceso del objeto o arma que ha sido recogido, para que "desaparezca" temporalmente.

Actualización del aiming: son sólo tres líneas que registran el movimiento del ratón para mover el centro de visión. En la última línea, se guarda el ángulo con el que se apunta en "oldangle" (para cuando el proceso estuviera "volando", ya que tiene dos ángulos, uno el ángulo de dirección en que se avanza y otro el ángulo en el que se mira).

Control del visor: cuatro escasas líneas para detectar la pulsación



Visión espléndida del magnánimo búnker.

de espacio y llamar al visor, explicado más adelante.

Gestor de la representación de armas en pantalla: es una parte bastante larga y que tampoco tiene tanto misterio, pero sería extenderse demasiado. Os aconsejo que lo miréis en el código fuente, ya que no es tan difícil. De todos modos he considerado esta parte como un "adorno" o "retoque profesional" al juego, ya que el modo de cambiar las armas, tan suave, y el balanceo de las mismas, no son estrictamente necesarios para que el juego funcione).

Control de disparo: está mezclado con la parte anterior porque es algo muy parecido, sólo que además de representar el arma llama a la función disparo con los parámetros pertinentes, indicando primero el tipo de disparo (0=láser de impulsos, 1=qk22/ADC, 2=bláster de fusión, 3=repetidor láser, 4=lanzamisiles, 5=mina).

Comprobación del máximo de municiones: a veces se supera el máximo de municiones, por ejemplo si se tienen 149 balas y se coge un cargador, no deberían aumentar a 159 sino a 150; en realidad aumentan a 159 pero esta parte lo corrige después, reduciéndolas a 150.

Control de la muerte: es una parte que, aunque funciona, no se puede usar todavía, así que lo dejaré para el próximo número) paciencia.

Restauración del ratón y caída en gravedad: el ratón se vuelve a centrar en pantalla y se cae (o se sube) la gravedad.

Con esto acaba la visión global del proceso que controla al jugador. ¿A qué no era tan complicado como parecía?

El visor

¿Qué es el visor? Os hago una pequeña introducción... en Div Deathmaker se puede escoger un skin para cada bot, pero si sólo se tiene un skin o poca memoria, es casi seguro que exista más de un bot con el mismo skin. ¡Vaya! ¿Cómo diferenciarlos al jugar con dos iguales? Con el visor. Éste es un proceso que manda una "sonda" en la dirección en que está mirando el jugador y al acercarse a un bot retorna su



Lanzamisiles y, enfrente, Quad Damage. Como podéis comprobar los mensajes no se cortan un pelo.

nombre. Es un proceso lento, por lo que sólo se activa al pulsar la tecla ESPACIO. Y es el único modo de diferenciar bots con el mismo skin, además de por su comportamiento.

En el PRG ya se avisa que visor es un proceso inestable debido a los propios bugs del modo8 y no se aconseja usarlo en exceso... por desgracia no puedo hacer nada para solucionarlo.

Los objetos y las armas

Los procesos que controlan los objetos y las armas son procesos muy simples y fácilmente entendibles. Simplemente, al ser llamados por *portalmuerte()*, se teletransportan al número de bandera que se le introduce como parámetro. Si no existe, desaparecen inmediatamente (porque no se definió esa bandera). En caso de existir la bandera "entran" en el modo8 y permanecen ahí. Aprovechan su variable local ANGLE para saber cuando han sido cogidas y cuando no. ANGLE es un contador que va aumentando a cada FRAME. Cuando un bot o el jugador coge el objeto (las armas no, porque siempre se pueden recoger si no se tienen ya) su ANGLE se reinicia, permaneciendo "escondidas" hasta que ANGLE alcanza su valor de contador y reaparecen, con un efecto de "reaparición" mediante procesos dot.

Los procesos DOT

Ya han aparecido mencionados antes estos procesos. ¿De qué se trata? Aquéllos que sepan qué significa dot pueden haberlo adivinado. Dot, en inglés, significa "punto". Son procesos que muestran un gráfico de UN PÍXEL, moviéndose en una dirección aleatoria en el mapa 3D. Sirven para efectos de teletransporte, choque de bala, impacto, sangre, explosión, etc... si habéis jugado a algún juego como Quake, que usa este efecto de "lanzar pixeles" usándolos como sprites pequeños, entenderéis mejor la utilidad de los procesos dot. Si no, podéis probar a coger un objeto (que no sea el Quad Damage, que tarda el doble en reaparecer) y quedaros mirando en dirección a donde estaba. Cuando aparezca veréis algunos destellos aparecer... son procesos dot.

Tabla 1: Estructura del proceso "jugador()"

Preparación del proceso, teletransporte de inicio.

COMIENZO DEL BUCLE:
Muestra en pantalla de munición, vida, blindaje.
Restauración de la pantalla (si había FADE).
Gestión del QUAD DAMAGE.
Gestión de la gravedad (parte importante).
Control del salto.
Control de pulsación de teclas 1-7 (para las armas).
Control de dirección (importante también).
Comprobación de colisiones con objetos y armas.
Actualización del "aiming" (dirección en que se apunta).
Control del "visor" (comprobador de bots, explicado en sección "El visor").
Gestor de la representación de armas en pantalla:

- Punto 1: muestra del cambio de arma.
- Punto 2: muestra del "balanceo" (efecto de movimiento).
- Punto 3 –externo a esta parte–: control del disparo (parte importante).

Intercambio de armas al acabarse la munición en una.

- Punto 4: muestra/control de la Qk22 cambiando de cargador.
- Punto 5: muestra del arma disparando.

Comprobación de que las municiones no superan los máximos.
Control de la muerte del jugador (implementado pero no utilizable todavía).
Restauración del ratón y caída en gravedad.
FIN DEL BUCLE Y DEL PROCESO

Apurando la memoria

Div Deathmaker no es un programa con requerimientos muy elevados. De hecho, el ordenador en que yo mismo lo programo es un P150 con 16 Mb de RAM, y eso tiene su parte buena: no puedo hacer un juego que no me funcione a mí mismo. Como algunos sabréis y habréis vivido ya, los programadores con recursos altos (Pc's muy potentes) tienden a desperdiciar memoria (en parte por "vaguería", en parte porque al no leer "se ha quedado sin memoria", no saben que están desaprovechándola)... Div Deathmaker funciona perfectamente con 16 Mb de RAM. Pero sólo una vez. Esto es confuso, pero tiene su explicación:

La memoria anda justita en el juego porque requiere gran cantidad de gráficos y Div consume muchos recursos. Pero hay un problema, la primera vez que se inicia una partida todo va de maravilla, pero una vez se termina y se quiere iniciar otra da un problema de falta de memoria (sólo con 16 Mb de RAM)... debido a LOAD_WLD. Esta función no dispone de una UNLOAD_WLD, y no es posible descargar un mapa 3D cargado con anterioridad, con lo que ahí radica el principal problema de pérdida de memoria en los juegos con modo8. Es un obstáculo irremediable y al que nos tendremos que enfrentar siempre.

¡Queremos jugar ya!

Esa es una de las frases con las que me encuentro a menudo. He intentado acelerar bastante esta parte, porque no es muy complicada, y dejar para el próximo número algo tan "relativamente" simple como los disparos y las explosiones (y sus respectivas interacciones con el jugador y los bots), el "tablón de puntuaciones" (accesible con una rápida pulsación de TAB) y la intrigante y querida IA de los bots, que tanto estáis esperando. No puedo asegurar que lleguemos a ver toda la IA en un solo número, pero procuraré que todo lo que esté en el próximo número sea ya jugable, aunque con carencias.

Y por supuesto, no me he olvidado de los sonidos y la música. Pero eso, como es lo habitual, lo dejamos para lo último.

Recordad que si tenéis alguna sugerencia/duda/o lo que sea me lo podéis mandar a mi e-mail las 24 horas del día los 365 días del año. ¡Hasta el próximo número!

Ferminho (Fermín Vicente)
ferminho@lycosmail.com

Programación de Juegos de Estrategia – 9

Principios básicos de IA (II)

Como decíamos el mes pasado, ahora iremos desarrollando las bases del sistema controlador; éste se encarga de manejar las unidades como si fuese un jugador humano enfrente del ordenador.

Es decir, es el que le dice a las unidades los objetivos que deben atacar, cuándo y cómo hacerlo. También es el que construye edificios en función de las necesidades de recursos. Una de las cosas que debe hacer el sistema controlador es construir los edificios que vaya necesitando. Esto significa que debe tener una política de prioridades. Si deseamos un adversario principalmente defensivo, concentrará sus esfuerzos en reforzar su terreno y cuando lo haya hecho, será cuando empiece a organizar una ofensiva. Si, por el contrario, deseamos que sea un adversario ofensivo, deberá construir muchas barracas para poder crear muchas unidades en poco tiempo. Tan sólo dedicará un pequeño porcentaje del tiempo y de los recursos a reforzarse.



Los naranjas están en inferioridad porque una de sus unidades (la roja) está demasiado lejos.

Control de las acciones realizadas

Para poder hacer que el ordenador reparta sus recursos en defensa o ataque, debemos conocer ciertos parámetros. En primer lugar, hay que saber los parámetros correspondientes a los recursos. Deberemos fijar una medida de tiempo para poder calcular los porcentajes que dedicados a cada actividad. A partir de ahora, llamaremos UT a la unidad de tiempo (1 UT puede ser por ejemplo equivalente a 10 FRAMES). Debemos tener una tabla en la que se guarden las UT's utilizadas en cada actividad, por ejemplo, construyendo o investigando. También nos sirve para calcular los ingresos de algún recurso por cada UT, porque saber que tenemos X unidades de oro si no sabemos cuánto estamos ganando cada cierto tiempo, no nos indica gran cosa. Si conocemos cuánto oro obtenemos por cada UT, podremos limitar la producción de unidades en función de su coste, o si lo que queremos es construir más unidades, podemos saber cuántas UT's necesitaremos o, podremos crear más fuentes de ingresos si los consideramos necesarios.

Otra utilidad para medir el tiempo es la posibilidad de hacer que ciertos eventos ocurran al rato de empezar a jugar o con cierta periodicidad durante el juego. Resumiendo, el ordenador únicamente puede procesar información, por lo que cuánta más le proporcionemos para procesar, más "inteligente" será nuestro controlador.

Cómo manipular la información obtenida

De momento, conocemos la posición en el espacio, la posición en el tiempo y el estado (andando, huyendo, espiando, etc.) en el que se encuentra cada unidad, así como la posición de cada una (tanto amigas como enemigas) y su cercanía (dentro de su campo de visión). Además, conocemos ciertos datos sobre cada unidad, como puede ser la vida que les queda. Con estos datos, podemos calcular el poder de una unidad, gracias a lo cual podremos comparar dos unidades entre sí.

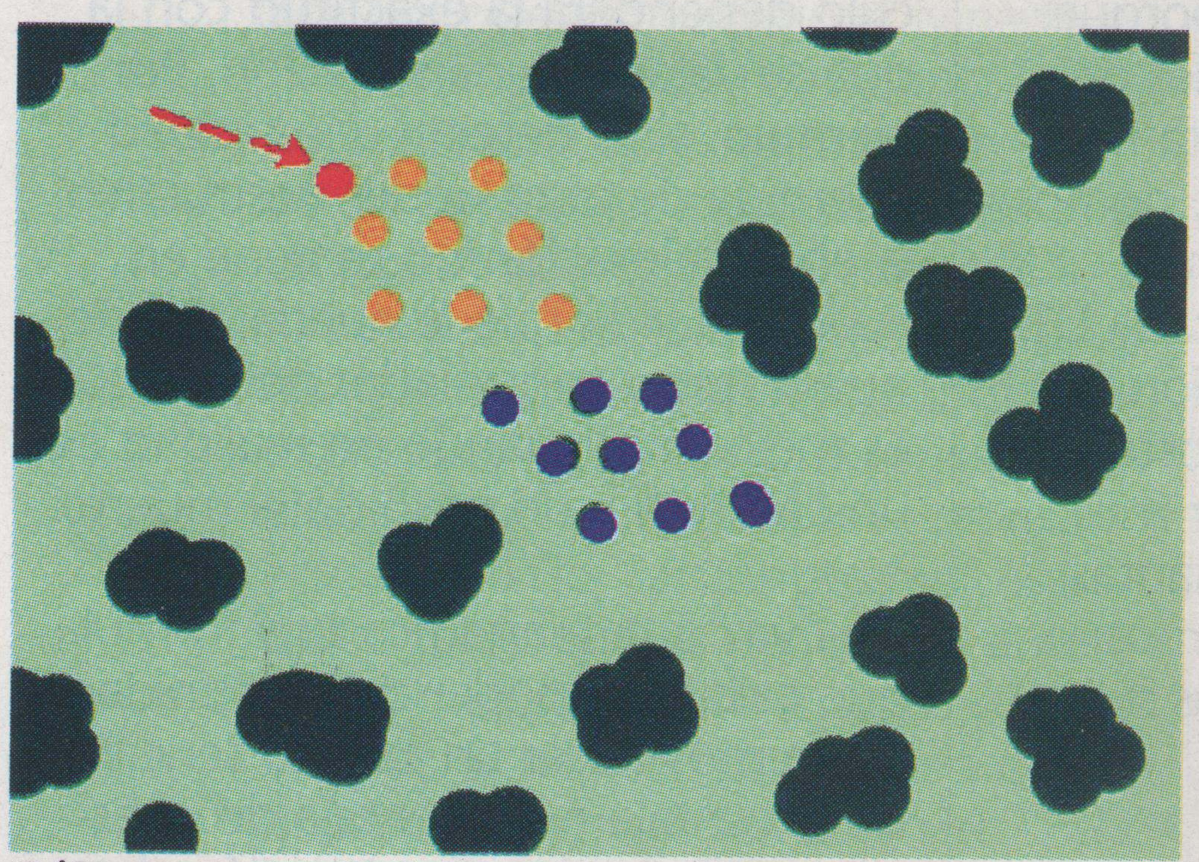
Este cálculo puede realizarse de muchas maneras, dando más importancia a unos valores e ignorando a otros. De la complejidad de este cálculo dependerá la exactitud con la que se valorará una situación. Por ejemplo, si se detecta una unidad enemiga de gran poder destructivo y, en la valoración no se tiene en cuenta la vida que le queda, puede que nuestro controlador decida que es mejor huir cuando en realidad esa unidad está a un paso de la muerte.

Una vez que tengamos una fórmula que calcule el poder de una unidad, podremos hacer un cálculo sencillo de la ventaja o desventaja de nuestras unidades en una situación determinada con tan sólo sumar las valo-

raciones de las unidades propias y restarle las valoraciones enemigas. Si el resultado es positivo, es que tenemos ventaja (cuanto mayor sea el número, mayor ventaja tendremos) y si es negativo, es que tenemos desventaja. Si queremos obtener una medida porcentual de la ventaja, sólo habrá que dividir las sumas de poderes, en vez de restarlas. Si el resultado es mayor de uno tendremos ventaja, en caso contrario, tendremos desventaja. La diferencia estriba en que con la ventaja porcentual, podemos poner un límite a la decisión de atacar independientemente de la suma total de poderes. Por ejemplo, si nos decidimos a atacar, siempre que poseamos una ventaja superior al 25%, lo haremos tanto si somos 126 contra 100, como si somos 126000 contra 100000. En ambos casos, la ventaja es del 26%, pero si el cálculo fuese absoluto, en un caso tendríamos una ventaja de 26 y en el otro de 26000. Según la situación, podremos utilizar una u otra fórmula.

La importancia del cálculo del poder de una unidad

La parte más importante para que este sistema funcione es calcular el poder de una unidad en una situación. Hacerlo demasiado simple puede hacer que el adversario parezca un idiota o un genio. Para poder calcular correctamente el poder de una unidad debemos pensar qué parámetros de los que conocemos afectan a la valoración de la unidad y si afectan de forma directa o inversamente proporcional. Algunos valores que afectan a la valoración de una unidad de forma positiva son la vida actual, la armadura, el poder de ataque, el daño medio que suele infligir, la velocidad, etc. En cambio, afectan negativamente la distancia real hasta poder atacar el objetivo, la separación con respecto al resto de las unidades de mi bando, el terreno, etc. Se puede observar que normalmente los datos independientes de la situación aumentan la valoración, mientras que los dependientes de la situación lo disminuyen.



Sólo acercando la unidad roja a la zona del conflicto mejoramos la situación.

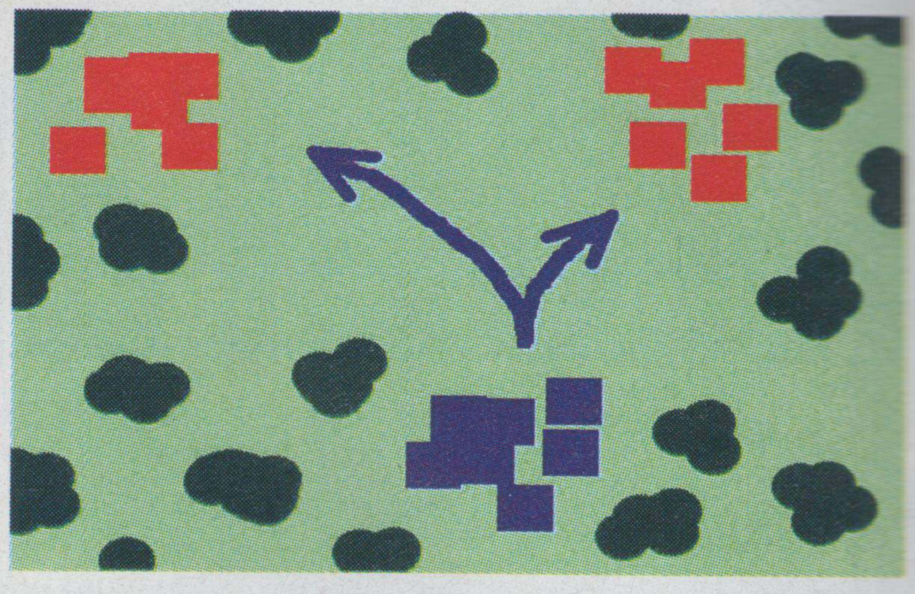
Puede que si en una situación en la no tenemos ventaja modificamos la posición de las unidades, acabemos teniendo ventaja, por lo que es bueno saber cuál es la valoración real y cuál la dependiente de la situación, para saber si podemos mejorar nuestra situación o, si por el contrario, ya está casi optimizada. Conocer los parámetros variables de la formula nos puede ayudar a saber qué es lo que debemos hacer para mejorar nuestra situación. De todas formas, aunque este sistema permite una adaptación al medio relativamente sencilla, no nos ayuda a crear un ejército equilibrado, sino a cómo colocarlo una vez que lo tenemos.

Cómo crear un ejército equilibrado

Para poder saber qué construir y cuándo, debemos preguntarnos lo que solemos hacer como jugadores. Normalmente, se construyen unidades que permiten obtener los recursos necesarios para seguir construyendo otras y luego, se construyen edificaciones que nos permiten construir otros tipos de unidades. Se deben levantar al menos una de cada tipo de unidad, siendo aconsejable construir dos o más de los edificios que nos permiten crear las unidades más comunes y baratas. No debemos construir demasiado lejos del núcleo de población, para que la protección sea más sencilla, ya que programar un controlador es más complicado, sobre todo cuando existe un número elevado de secciones.

Podría ocurrir que para defender una sección se considerase necesario (por los ajustes de poder) enviar las unidades de otra sección, por lo que ésta segunda quedaría desprotegida. Es cierto que esta situación se puede prever, pero es más sencillo limitar la construcción a una determinada zona de terreno, que calcular todas las posibles situaciones. Si se desean varios núcleos, lo mejor es tener varios bandos, cada uno con sus unidades y edificios, como sucede en la mayoría de los juegos de estrategia. Así, cada bando es controlado por un controlador, lo que evita los conflictos.

Siguiendo con el tema de las unidades, lo mejor es predefinir varias tablas con proporciones de los tipos de unidades, según la estrategia a seguir o el enemigo a vencer. Siempre deberemos considerar el ejército como dos partes: la atacante,



Los azules pueden atacar a uno u otro sin que el contrario intervenga para no quedar desprotegido. Por eso no conviene dividirnos.

y la defensiva. Hasta que esta última esté constituida, no deberemos gastar demasiados recursos en la parte atacante. Una vez que la defensa esté constituida, tan sólo deberemos mantenerla, reconstruyendo lo destruido y reponiendo las unidades perdidas. En cambio, el resto de recursos se utilizarán para formar grupos de ataque de mayor o menor potencia. Si nos fijamos en otros juegos del género, el enemigo realiza un ataque cada cierto tiempo con las unidades que haya conseguido producir. De esta forma, se impide al enemigo formar una defensa perfecta y, se le producen bajas que ha de reponer, por lo que no podrá dedicarse de lleno a su objetivo. Se deberá fijar el porcentaje de los recursos que se va a utilizar en la investigación de nuevos tipos de unidades o para mejorar las existentes, pero al ser algo tan subjetivo, tan sólo recordaremos que se debe hacer.

La ventaja de la información

Como decíamos antes, el ordenador basa sus acciones en la información que posee del enemigo, por lo que cuánta más información posea, más inteligente parecerá el adversario. Ésta es una forma de que sea un adversario más difícil, por lo tanto, se le da cierta ventaja sobre su oponente humano, que consiste en que debe conocer todo lo que ocurre en el mapa, aunque no lo vea. Esto le permite preparar estrategias de ataque que de otra forma no podría, pero también puede hacer que las tácticas del jugador no resulten tan efectivas como lo serían contra un jugador humano real. Como segunda opción para mejorar las prestaciones del ordenador, se le pueden dar otras ventajas, además de la información. Un mayor número de unidades o mayores fuentes de recursos, combinados con un pequeño aumento de la velocidad de las unidades recolectoras de recursos, puede hacer que el ordenador resulte un adversario terrible.

Emilio Llamas Alba (YuMoK)

Curso de juegos de rol



Vistiendo y equipando personajes

En el artículo de Divmanía de este mes veremos cómo implementar fácilmente una interfaz gráfica para poder vestir y equipar a los personajes que vayamos metiendo en nuestro juego de rol.

El uso de una interfaz gráfica da una mayor vistosidad al juego y de hecho, todos los juegos de rol actuales las usan.

Las interfaces gráficas, pese a no ser demasiado complejas, presentan ciertos aspectos que las hacen algo más difíciles de hacer que las convencionales interfaces en modo texto. Así que, para ver cómo se podría realizar dicha interfaz gráfica, estudiaremos el siguiente código:

```
Program equipo;

Const

    n_equipado=2;
    n_equipable=5;
    distx=20;
    disty=20;
    startx=250;
    starty=30;

Global

    struct objetos[6]
        string nombre;
        int atack;
        int defense;
        int graph;
        int tipo;
    end

    struct equipadoxy[2]
        int x;
        int y;
    end

    int equipado[n_equipado];
    int equipable[n_equipable];

    int fpg;

Begin

// DEFINIMOS LOS OBJETOS

    objetos[0].nombre="Espada";
    objetos[0].atack=10;
    objetos[0].defense=0;
    objetos[0].graph=1;
    objetos[0].tipo=0;

    objetos[1].nombre="Katana";
    objetos[1].atack=15;
    objetos[1].defense=0;
    objetos[1].graph=2;
    objetos[1].tipo=0;

    objetos[2].nombre="Escudo
peque¤o";
    objetos[2].atack=0;
    objetos[2].defense=10;
    objetos[2].graph=3;
    objetos[2].tipo=1;

    objetos[3].nombre="Escudo gran-
de";
    objetos[3].atack=0;
    objetos[3].defense=15;
    objetos[3].graph=4;
    objetos[3].tipo=1;

    objetos[4].nombre="Armadura de
cuero";
    objetos[4].atack=0;
    objetos[4].defense=20;
    objetos[4].graph=5;
    objetos[4].tipo=2;

    objetos[5].nombre="Armadura de
hierro";
    objetos[5].atack=0;
    objetos[5].defense=30;
    objetos[5].graph=6;
    objetos[5].tipo=2;

    objetos[6].nombre="";
    objetos[6].atack=0;
    objetos[6].defense=0;
    objetos[6].graph=7;

// RELLENAMOS LA ESTRUCTURA
CON LAS COORDENADAS DE LOS
OBJEOS A EQUIPAR

    equipadoxy[0].x=50;
    equipadoxy[0].y=105;

    equipadoxy[1].x=90;
```



```
    equipadoxy[1].y=105;

    equipadoxy[2].x=130;
    equipadoxy[2].y=105;


  // RELLENAMOS LA TABLA EQUIPA-
  BLE CON LOS OBJETOS QUE QUERE-
  MOS

    equipable[0]=0;
    equipable[1]=1;
    equipable[2]=2;
    equipable[3]=3;
    equipable[4]=4;
    equipable[5]=5;

  // OBJETOS EQUIPADOS

    equipado[0]=6;
    equipado[1]=6;
    equipado[2]=6;

  // CARGA DE FICHEROS

    fpg=load_fpg("equipo.fpg");

  // CAMBIAMOS LA RESOLUCION

    set_mode(m320x240);

  // PONIENDO EL FONDO DE PANTA-
  LLA

    put_screen(fpg,20);

  // DEFINIMOS LA REGIÓN QUE
  CONTENDRA LOS OBJETOS EQUIPA-
  BLES Y DIBUJAMOS LAS LÍNEAS PARA
  MARCAR LAS CASILLAS

    define_region(1,startx-
  (distx/2),starty-
  (disty/2),distx*2,disty*((n_equipa-
  ble+1)/2));
    draw(2,34,10,0,240,20,280,80);
    draw(1,34,10,0,260,20,260,80);
    draw(1,34,10,0,240,40,280,40);
    draw(1,34,10,0,240,60,280,60);

  // DEFINIMOS LAS REGIONES DE
  LOS SITIOS DONDE PODEMOS
  EQUIPAR COSAS
    define_region(2,40,95,20,20);
```



```
    draw(2,34,10,0,40,95,60,115);

    define_region(3,80,95,20,20);
    draw(2,34,10,0,80,95,100,115);

    define_region(4,120,95,20,20);

draw(2,34,10,0,120,95,140,115);

    equipar();


    Loop
      dibujar();

      Frame;
    End


End

///////////////////////////////////////
//////////////////////////

Process equipar()

Private

  byte pos;
  int cobj;
  int cpos;
Begin
  cpos=-1;
  graph=21;
  Loop
    x=mouse.x;
    y=mouse.y;
    If(mouse.right)
      if(!out_region(id,1))

pos=halla_pos(mouse.x,mouse.y);

info_object(equipable[pos]);
      else
        if(!out_region(id,2))
          pos=0;
        end

        if(!out_region(id,3))
          pos=1;
        end

        if(!out_region(id,4))
          pos=2;
        end


info_object(equipado[pos]);
        end

      End

      If(mouse.left )

        If(!out_region(id,1))

pos=halla_pos(mouse.x,mouse.y);
          if(equipable[pos]!=6)
            if(cpos<0)
```



```
graph=objetos[equipable[pos]].graph
;

cobj=equipable[pos];
              equipable[pos]=6;
              cpos=pos;
            else
              graph=21;

equipable[cpos]=cobj;
              cpos=-1;
            end
          end
        else
          If(!out_region(id,2) &&
cpos>-1 && objetos[cobj].tipo==0)
            if(equipado[0]==6)
              equipado[0]=cobj;
              cpos=-1;
              graph=21;
            else

equipable[cpos]=equipado[0];
              equipado[0]=cobj;
              cpos=-1;
              graph=21;
            End
          End

          If(!out_region(id,3) &&
cpos>-1 && objetos[cobj].tipo==2)
            if(equipado[1]==6)
              equipado[1]=cobj;
              cpos=-1;
              graph=21;
            else

equipable[cpos]=equipado[1];
              equipado[1]=cobj;
              cpos=-1;
              graph=21;
            End
          End

          If(!out_region(id,4) &&
cpos>-1 && objetos[cobj].tipo==1)

            if(equipado[2]==6)
              equipado[2]=cobj;
              cpos=-1;
              graph=21;
            else

equipable[cpos]=equipado[2];
              equipado[2]=cobj;
              cpos=-1;
              graph=21;
            End
          End
```

```
if(cpos>-1)
        graph=21;

equipable[cpos]=cobj;
        cpos=-1;
    end


        End
    End

        Frame;
    End
End

//////////////////////////////////////
//////////////////////////////////////
///

Function halla_pos(xm,ym)

Private
    int xr;
    int yr;
    byte rx;
    byte ry;
    byte nr;
Begin
    xr=xm-startx+10;
    yr=ym-starty+10;
    rx=xr/distx;
    ry=yr/disty;
    nr=ry*2+rx;
    return(nr);
End

//////////////////////////////////////
//////////////////////////////////////
///

Function dibujar()

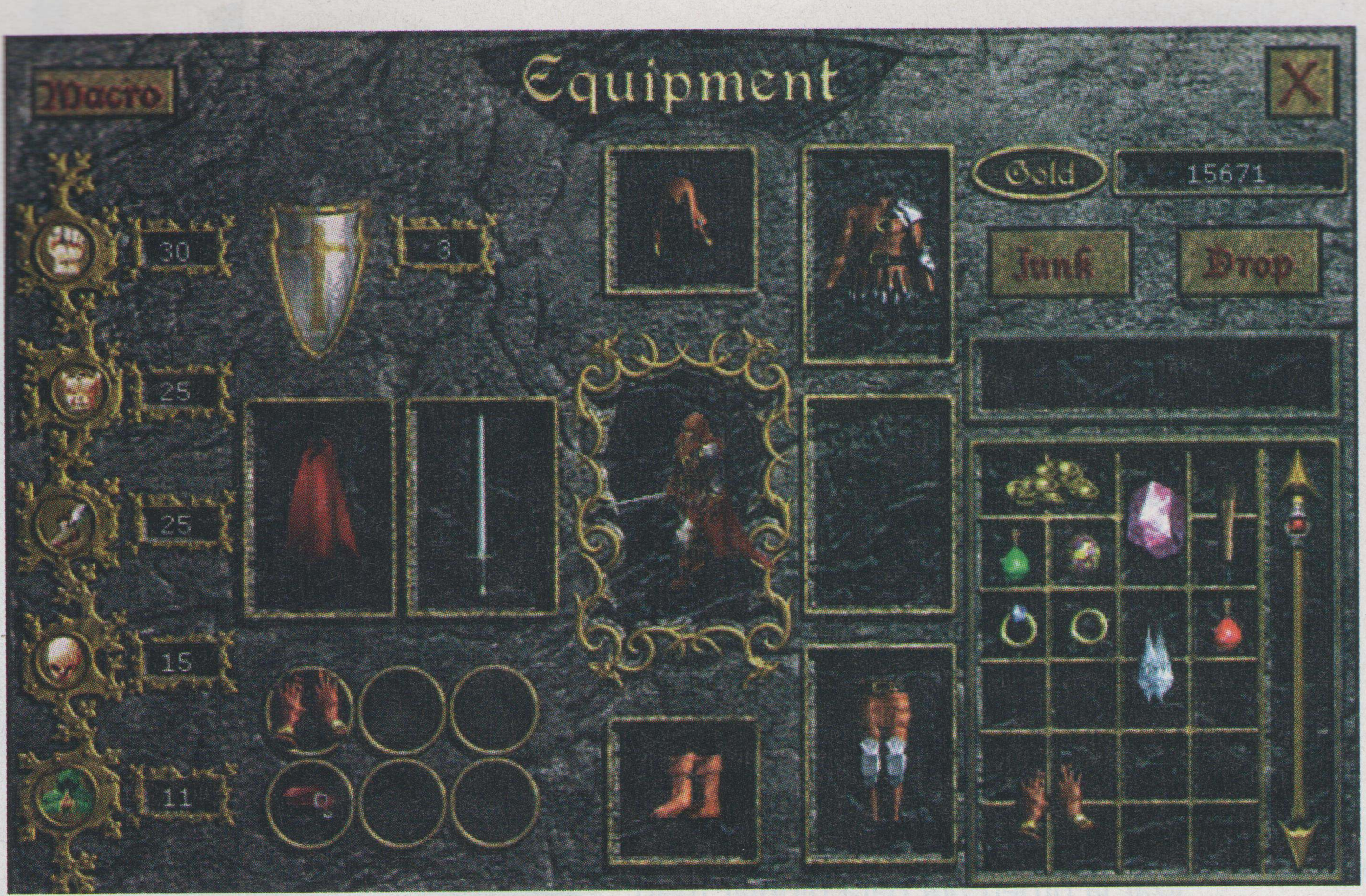
Private

    byte i;
    int temp_map;

Begin

temp_map=new_map(320,240,160,
120,0);
```





Esta es la interfaz gráfica que usa T4C (The 4th coming) para equipar a nuestro personaje.

```
    For(i=0;i=<n_equipable;i+=2)

map_put(fpg,temp_map,objetos[equi
pable[i]].graph,startx,starty+((disty/
2)*i));
    End

    For(i=1;i<=n_equipable;i+=2)

map_put(fpg,temp_map,objetos[equi
pable[i]].graph,startx+distx,starty+((
disty/2)*(i-1)));
    End

    For(i=0;i<=n_equipado;i++)

map_put(fpg,temp_map,objetos[equi
pado[i]].graph,equipadoxy[i].x,equi-
padoxy[i].y);
    End

    put(0,temp_map,160,120);

End

//////////////////////////////////////
//////////////////////////////////

Function info_object(obj)

Private

    int idtext;

Begin

    delete_text(idtext);
    idtext=write(0,200,200,0,obje-
tos[obj].nombre);

    write(0,200,210,0,"Ataque:");

write_int(0,250,210,0,&objetos[obj].
atack);
    write(0,200,220,0,"Defensa:");

write_int(0,250,220,0,&objetos[obj].
defense);


End
```

Cómo definimos los objetos y las tablas de los arrays que los contendrán

El programa se basa es dos tablas. La tabla *equipados* y la tabla *equipables*. Estas dos tablas contienen un número que indica el tipo de objeto que contienen.

Por ejemplo, *equipados[2]=3* indicaría que en la tercera posición del array encontraremos un objeto de tipo 3.

Los objetos vienen definidos en el array de estructuras *objetos*. En dicho array de estructuras se define el nombre y características que el objeto en cuestión posee. El array de estructuras objetos se define de la siguiente manera:

```
struct objetos[6]
    string nombre;
    int atack;
    int defense;
    int graph;
    int tipo;
end
```

Dentro de la estructura tenemos el nombre del objeto, su ataque, defensa, gráfico y tipo. La variable tipo es usada para distinguir los objetos, ya que, por ejemplo, no es lo mismo una espada que una armadura. La variable graph asocia al objeto el código de un gráfico.

Cómo dibujaremos los gráficos

Dado que son bastantes los objetos que aparecerán dibujados a la vez, no sería demasiado correcto hacer un proceso para cada uno, por lo que habrá que usar un método distinto para dibujarlos.

Para dibujar todos los gráficos, primero crearemos un mapa temporal cuyas dimensiones coincidirán con el alto y ancho de la pantalla (en nuestro caso 320x240). Después, mediante la función *map_put* iremos poniendo los gráficos uno a uno en el mapa temporal que anteriormente hemos creado.

Una vez hemos puesto todos los gráficos tan solo nos hace falta utilizar la orden *put* para dibujar todo el mapa.

La estructura de la función de dibujado sería la siguiente:

```
temp_map=new_map(ancho,alto,ancho/2,alto/2,color);

    For(i=0;i=<num_obj;i+=2)
        map_put(fpg,destino,origen,x,y);
    End

put(0,temp_map,ancho/2,alto/2);
```

Cómo averiguar qué objeto se encuentra en unas coordenadas (x,y)

Para averiguar qué objeto se encuentra en una determinada posición utilizaremos la función *halla_pos*, que es la siguiente:

```
Function halla_pos(xm,ym)

Private
    int xr;
    int yr;
    byte rx;
    byte ry;
    byte nr;
```





Captura del menú de equipo de Asheron's Answer.

```
Begin
    xr=xm-startx+10;
    yr=ym-starty+10;
    rx=xr/distx;
    ry=yr/disty;
    nr=ry*2+rx;
    return(nr);
End
```

Esta función recibe como parámetros la *x* e *y* donde se pulsa uno de los botones del mouse.

Una vez ha recibido los parámetros, se halla la posición relativa de ese punto respecto al (0,0). Después se busca la región en la que se encuentra (dividimos la zona donde están los objetos en casillas de 20x20 píxels) dividiendo los valores de la *x* y la *y* por el largo y ancho de las casillas.

Una vez ya sabemos la casilla donde se encuentra, encontramos qué objeto es mediante *nr=ry*2+rx*, ya que el número de casillas horizontales siempre es 2.

Uso de las regiones

En este ejemplo se hace un uso extensivo de las regiones, que son de gran utilidad para determinar si un proceso se encuentra en una zona rectangular previamente determinada.

Para usar una región en DIV, primero se ha de crear la región con la función:

define_region(<numero de región>,x,y,ancho,alto);

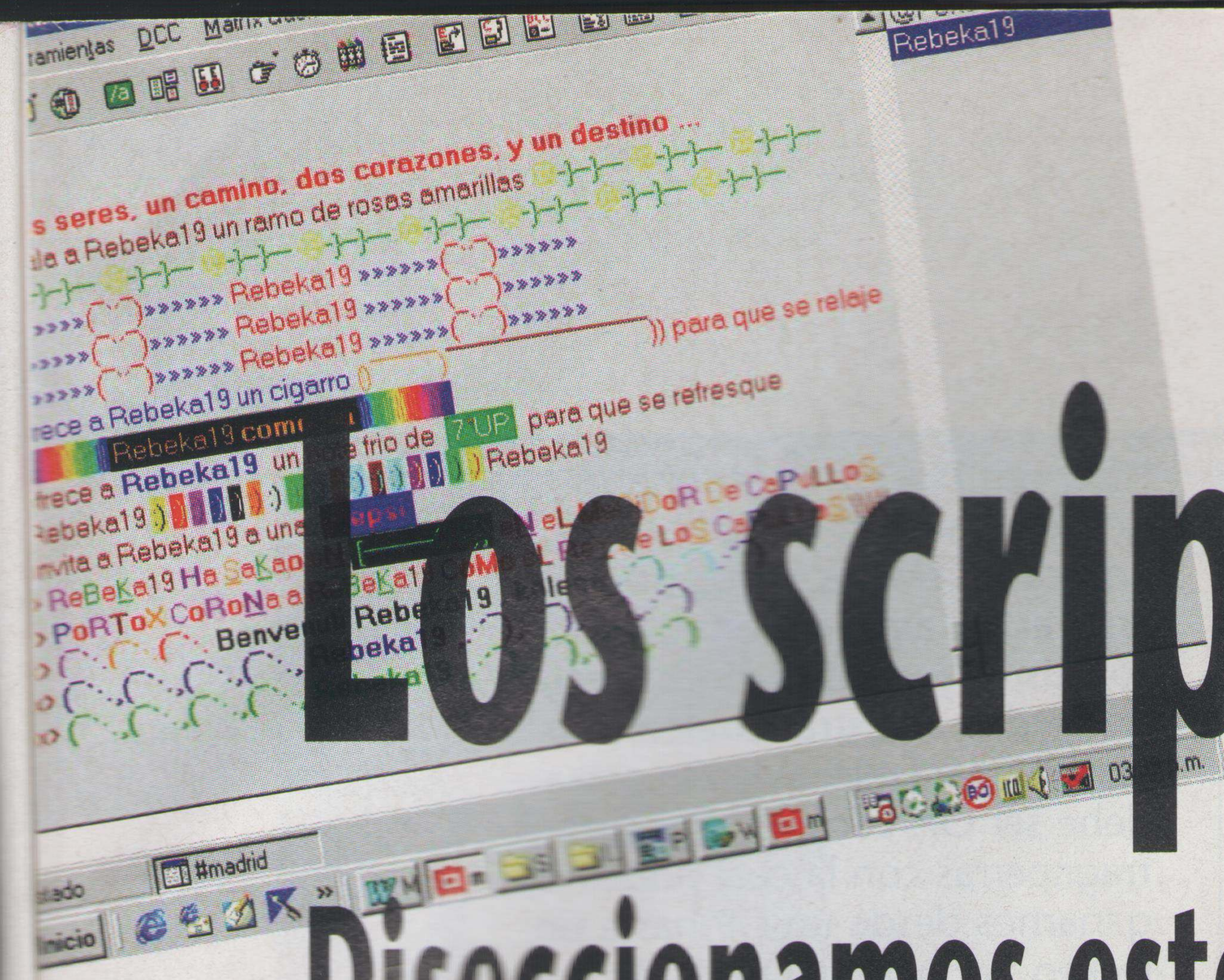
Una vez que la región está definida, tan sólo hemos de usar la función:

out_region(<id de proceso>,<numero de region>);

para comprobar si un proceso está dentro o fuera de la región.

En nuestro ejemplo, se usa para saber si el proceso a equipar se encuentra en una determinada zona y así poder actuar en consecuencia. Por ejemplo, si se encuentra en la región 1 sabemos que se trata del lugar donde están los objetos equipables, si es la región 2 es donde equiparemos la espada, etc.

Ramón de España
maqnaziller@pagina.de



Los scripts

Diseccionamos estos programas

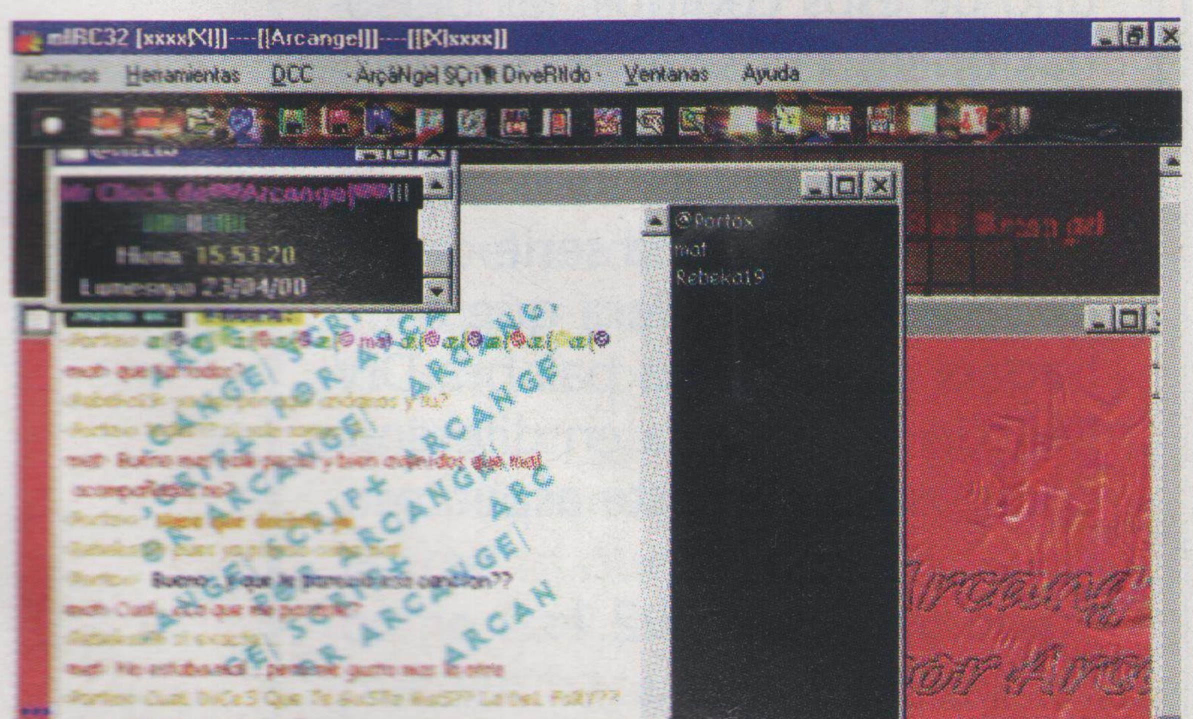
Los Scripts son programas para el IRC (Internet Relay Chat), es decir, que sirven para chatear. Estos Scripts están basados principalmente en un programa, el famoso mIRC, a partir de él y mediante una serie de menús llamados Alias, Popus y Remotos que se pueden crear fácilmente, surge lo que se conoce como Script.

Resumiendo, que un Script es el programa mIRC modificado por una o más personas, al añadirle nuevos menús y nuevas opciones.

Tipos de scripts

Los Scripts existentes actualmente, tanto en español como en castellano son muchos y muy variados, los hay que están orientados principalmente para atacar a otros usuarios del IRC molestándolos o desconectándolos, lo que se conoce como IRC WAR, los hay para defenderse de estos ataques, los hay orientados simplemente para hacernos más ameno el chateo incluyendo opciones muy interesantes como escribir con colorines, contar chistes, jugar a juegos, etc. También los hay orientados hacia principiantes, aunque la mayoría de los Scripts incluyen una mezcla de todas estas opciones.

> Los Scripts nos protegen del ataque de otros usuarios en lo que se ha llamado la IRC-WAR



Principales aplicaciones de los scripts

Es muy difícil ponerse de acuerdo en qué aplicaciones extras nos puede aportar un Script a nuestro chateo, ya que como dijimos antes eso depende mucho del tipo de Script, aunque intentaremos mostraros la mayoría de las aplicaciones extras que existen, un Script no tiene por que tenerlas todas.

Para empezar hay que decir que aunque el Script es una modificación del mIRC, incluye todas las opciones que trae de por sí este programa, lo que quiere decir que por lo pronto ya tenemos una gran lista de servidores donde conectarnos, la posibilidad de grabar nuestras conversaciones en ficheros de texto, más conocidos como Logs, la posibilidad de enviar y recibir archivos mediante DCC o de chatear por DCC evitando el fatídico LAG y muchas otras opciones más.

Pero, como muchas veces esas opciones no son suficientes, los Scripts incluyen muchas otras que hacen mas interesante y ameno el chateo, a continuación explicaré muchas de ellas:

- IRC-WAR: Como ya explicamos antes, esta opción sirve para poder atacar a los otros usuarios del IRC molestándolos o desconectándolos, raro es el Script que hoy en día no incluya al menos alguna que otra opción de IRC-WAR, dentro de la IRC-WAR existen muchos tipos de ataques, en los que no vamos a entrar a describir en este artículo, pero los mas conocidos son los famosos Nukes.
- PROTECCION: Al igual que existe la IRC-WAR también existen programas para protegerse de estos ataques, hoy en día todos los Scripts llevan protecciones de este tipo en mayor o menor número y eficacia, protecciones contra Nukes, Flood y otros tipos de ataques son casi imprescindibles para que nadie nos pueda molestar en el IRC.
- CLONES Y BOTS: Son copias de tu mIRC que cargas en el servidor y hacen que estés presente en el IRC varias veces. Se pueden usar para atacar a otros usuarios mediante flood o bien si tienes Op, puedes cargar un bot y hacerle Op, así si te tiran, aun conservarás una copia tuya con Op. Tradicionalmente los Clones son las copias que se usan en la IRC-WAR y los Bots son las que se usan para fines pacíficos.
- OPCIONES OP: El mIRC incluye de por sí ya opciones para cuando

mIRC

Es un programa muy utilizado por los internautas para chatear por la red debido a sus amplias posibilidades para modificarlo a nuestro gusto cambiando los menús, así como por su sencillez y rapidez. El programa lleva ya varias versiones y la primera salió a la luz en 1995. Si queréis mas información o bajaros el programa visitar http://www.mirc.co.uk y si queréis aprender a usarlo os aconsejo que visitéis:

http://www.readysoft.es/home/ihidalgo/contenido.html

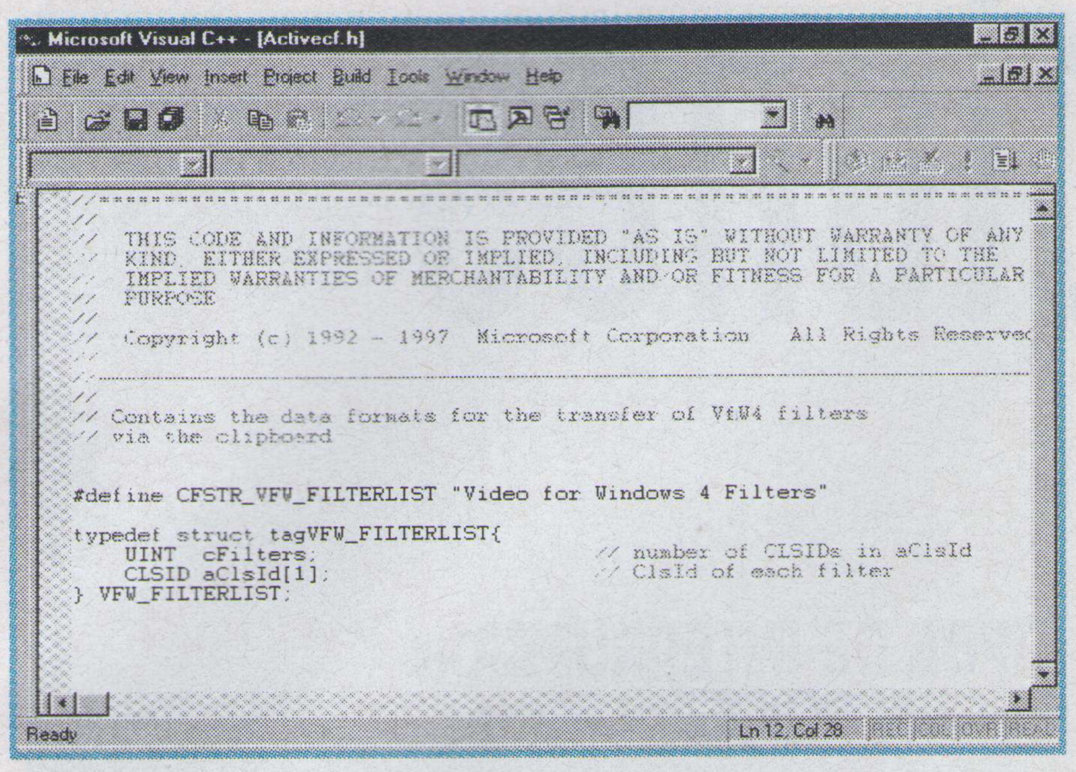


FIGURA 1.

No es una función muy útil: simplemente imprime los dos valores que se le pasan. Sin embargo, demuestra un punto importante: la función recibe los dos valores pasados por el programa y los almacena, o mejor, almacena duplicados de ellos en su espacio privado de memoria. De hecho, puede dar nombres diferentes a esos valores conocidos sólo por la función: en este caso, *xx* e *yy*, en lugar de *x* e *y*. La función puede manejar las nuevas variables, *xx* e *yy*, sin afectar a las originales, *x* e *y*, del programa.

Paso de direcciones a una función

Veamos ahora el caso contrario: pasar dos valores de la función al programa que la llamó. ¿Cómo podemos hacer esto? Se utiliza un proceso de dos escalones. Primero, el programa, en lugar de pasar valores a la función, pasa sus direcciones.
Éstas son las direcciones de las variables del programa donde se quieren almacenar los valores retornados. Veamos el programa:

```
/* direccio.c */
/* función de prueba que devuelve dos valores */
void dev2(int *, int *);

main( )
{
 int x, y;
   dev2(&x, &y);
  printf("El primero es %d, el segundo %d", x, y);
}

/* dev2( ) */
/* imprime dos números */
void dev2(int *px, int *py)
{
 *px = 3;
 *py = 5;
}
```

El programa tampoco realiza nada demasiado útil. El programa *main( )* llama a la función *dev2( )*, que devuelve dos valores, 3 y 5, al programa. Posteriormente, el programa imprime estos valores. A pesar de su aparente inutilidad, el listado está lleno de nuevas ideas. Veámoslas paso a paso.
Primero, observamos que el programa no da ningún valor a las variables *x* e *y*. Pero cuando se ejecuta el programa, estas variables tienen un valor, como podemos ver en la ejecución, ya que es el programa el que las imprime. Podemos inferir que la función *dev2( )* suministra, de alguna forma, estos valores al programa.
El programa le dijo a *dev2( )* dónde poner los valores al pasar las direcciones de las variables. La forma de conseguir esto es mediante el empleo del operador de dirección &. La expresión

Dev2(&x, &y);

pasa las direcciones de *x* e *y* a la función, que las almacena en su espacio privado de memoria. La función da nombre a estas direcciones: *px* y *py*. De esta forma, la función puede referirse a ellas, como si fueran otro tipo de variables (naturalmente podríamos haber utilizado otros nombres, por ejemplo *punt_a_x* y *punt_a_y*, que son más descriptivos pero más largos).

Declaración de variables de puntero

Como sucede con otros tipos de variables, se deben declarar los lugares reservados para estas direcciones, *px* y *py*, ya que el compilador debe saber sus nombres y cuánto espacio de memoria tiene que asignarles. Como estamos almacenando direcciones, o constantes de punteros, se supone que existe otro tipo de dato, diferente a los vistos hasta ahora. Algo como lo siguiente:
Ptr px, py

Siendo *ptr* el tipo de dato para los punteros. En general, todas las direcciones tienen el mismo tamaño y queremos reservar la memoria suficiente para almacenar una dirección. Normalmente, dos bytes son suficientes para almacenar una dirección (cuando se utilizan modelos de memoria distintos al modelo *pequeño*, esto puede dejar de cumplirse; sin embargo, todos los ejemplos contenidos en este cursillo utilizan el modelo pequeño).
Al declarar una variable de puntero reservamos dos bytes de memoria, pero existe una complejidad añadida. Por motivos que explicaremos más adelante, el compilador necesita conocer no sólo que hemos declarado un puntero, sino a qué tipo de dato está apuntando el puntero. En otras palabras, cada vez que reservamos espacio para almacenar la dirección de una variable, necesitamos decirle al compilador el tipo de dato de esta variable. Especulemos otra vez con la posible apariencia que tendría la declaración de un puntero:

Int _ptr px, py;

Siendo *int _ptr* el tipo de dato para los punteros que apuntan a variables enteras. Nos vamos acercando. Sin embargo, C es un lenguaje conciso; por tanto, en lugar de utilizar la palabra *ptr*, emplea el asterisco. El asterisco actúa como diferenciador de las palabras que representan tipos de datos simples (por ejemplo, *int* y *float*) y se utiliza inmediatamente antes de cada *variable*, en lugar de utilizarse una sola vez al principio de la declaración. De esta forma la declaración para dos punteros a enteros es:

*Int *px, *py;*

La declaración reserva dos bytes en los que se almacena la dirección de una variable entera y denomina a este espacio de almacenamiento con el nombre *px*. También reserva otros dos bytes en los que almacena la dirección de otra variable entera y le da el nombre de *py*. Los asteriscos dicen al compilador que estas variables contienen direcciones (no valores) y la palabra *int* indica que estas direcciones apuntan a variables enteras. Observamos que la declaración no dice nada sobre lo que se colocará en esas variables.

El estilo conciso de este formato de declaración es uno de los motivos de confusión con los punteros. Por tanto, reiteramos lo que ocurre: para cada variable (*px* y *py* en este caso) la declaración hace que el compilador reserve un espacio en memoria de dos bytes en los que se pueden colocar sus direcciones.
Además, el compilador es consciente del tipo de variable a la que se refiere la dirección, en este caso enteros.

Pasar valores a las variables de punteros

El programa llama a la función mediante la instrucción:

dev2(&x, &y);

las direcciones proporcionadas por el programa, *&x* y *&y*, se colocan en los espacios indicados en la declaración de la función. De esta forma, se pasa el control y las dos direcciones (que pueden ser 1310 y 1312) a la función. Las dos direcciones se colocan en el espacio reservado para *px* y *py*.

Examinemos este proceso cuidadosamente para asegurarnos que estamos en el buen camino. Podemos decir que *px* y *py* son variables de punteros y que las direcciones 1310 y 1312 son constantes de punteros.

CURSO DE ENSAMLADOR

Usando las interrupciones

Las interrupciones son parte constituyente del propio diseño del microprocesador y son las que hacen posible que funcione el teclado, el reloj interno, la BIOS y el propio sistema operativo. En este capítulo se va a detallar cómo poder acceder a esta completa colección de rutinas.

Las interrupciones, desde el punto de vista práctico del programador, son las rutinas que están instaladas en el sistema y que están a nuestra disposición como si de una gran librería de funciones residentes en memoria se tratara y las cuales podemos utilizar en cualquier momento desde nuestras aplicaciones.
Estas funciones se clasifican en grupos según su utilidad, teniendo, así, que disponemos de rutinas para escribir en pantalla, para leer y escribir en disco, para gestionar la memoria y otras funciones de mucha utilidad y que, en caso de tener que realizar nosotros mismos directamente mediante código *a mano*, haría que la programación en ensamblador fuese un verdadero infierno, ya que habría que conocer tan a fondo el hardware de la máquina como los propios creadores del mismo. Así, por ejemplo, simplemente para realizar la operación de crear un fichero en el disco, en caso de no disponer de rutinas destinadas a ello, habría que realizar un gigantesco programa que tendría que acceder directamente a nivel binario con la controladora de disco, para lo cual se necesitarían conocimientos completos del funcionamiento interno de la misma.

CLASIFICACION DE LAS INTERRUPCIONES

En total existen 256 interrupciones, que se numeran desde siempre por costumbre en hexadecimal (así lo veremos en todos los libros), por lo que hay interrupciones desde la 00h hasta la Ffh.
Estas 256 interrupciones están divididas en cuatro grandes grupos, que son: las interrupciones BIOS, las interrupciones DOS, las de usuario (libres) y las del BASIC (que llevaban los primeros ordenadores IBM PC en la ROM y el cual se ejecutaba automáticamente cuando no se encontraba un disco de arranque).
Cada uno de estos grupos mencionados se divide, a su vez, en subclases según su utilidad. En la **tabla 1** puede verse un mapa de todas las interrupciones.

TIPOS DE INTERRUPCION SEGUN ORIGEN

Independientemente de los cuatro grupos a los que pertenecen todas las interrupciones (BIOS, DOS, Usuario y BASIC), todas ellas se refieren, a su vez, a tres clases dependiendo de si son ejecutadas cuando lo pide la propia CPU, si lo pide un dispositivo físico externo o si sólo pueden ejecutarse explícitamente desde código de programa. Según lo dicho, se llamarán excepciones, interrupciones hardware o interrupciones software.
Las excepciones son aquellas interrupciones que se ejecutan cuando ocurren situaciones anómalas de funcionamiento de la CPU.

TABLA 1: MAPA DE INTERRUPCIONES DEL SISTEMA

Interrupciones BIOS:
00h- 07h: Interrupciones internas del microprocesador.
- 00h: División por cero.
- 01h: Ejecución paso a paso.
- 02h: No enmascarable.
- 03h: Breakpoint.
- 04h: Overflow.
- 05h: Imprimir pantalla.
- 06h-07h: No usadas.

08h- 0Fh: Interrupciones del controlador 8259.
- 08h: Temporizador.
- 09h: Teclado
- 0Ah-0Dh: No usadas.
- 0Eh: Diskette.
- 0Fh: No usada.

10h- 1Ah: Puntos de entradas de la BIOS.
- 10h: Entrada/Salida Pantalla.
- 11h: Chequeo del equipo físico.
- 12h: Tamaño de la memoria.
- 13h: Diskette.
- 14h: Entrada/Salida Puerto serie.
- 15h: Entrada/Salida cassette.
- 16h: Entrada/Salida Teclado.
- 17h: Entrada/Salida Impresora.
- 18h: Llamada al BASIC de cassete.
- 19h: Reinicializar el sistema.
- 1Ah:Contador del temporizador.

1Bh- 1Ch: Rutinas del usuario.
- 1Bh: Rutinas del teclado.
- 1Ch: 'Tic' del temporizador.
- 1Dh: Inicialización de vídeo.
- 1Eh: Parámetros de diskette.
- 1Fh: No se usa.

1Dh- 1Fh: Parámetros de la BIOS.

Interrupciones DOS:
20h: Terminación del programa.
21h: Petición de una función.
- Servicios disponibles:
- AH=0h a 12h: Entrada/Salida Carácter.
- AH=13h a 24h: Gestión de ficheros.
- AH=25h a 26h: Funciones varias.
- AH=27h a 29h: Gestión de ficheros.
- AH=2Ah a 2Eh: Funciones varias.
- AH=2Fh a 38h: Sólo a partir del DOS 2.0.
- AH=39h a 3Bh: Gestión de directorios.
- AH=3Ch a 46h: Gestión de ficheros

22h: Dirección de finalización del programa.
23h: Dirección de salida con CTRL+BREAK
24h: Rutina gestora de errores críticos.
25h: Lectura absoluta de disco.
26h: Grabación de disco absoluta.
27h: Terminar programa dejándolo residente.
28h- 3Fh: Reservadas, no usadas.

Interrupciones de Usuario:
40h- 7Fh: Libres para uso del programador.

Interrupciones B.A.S.I.C.
80h- Ffh: Funciones del BASIC
(si está presente)

TABLA 2: MAPA DE INTERRUPCIONES DEL SISTEMA

00h: Se ha producido una división por cero.
01h: Paso a paso. Cuando el modo depuración está activado se ejecuta tras cada instrucción.
03h: Punto de ruptura encontrado. Usado en modo de depuración.
06h: Se ha producido un Código de operación inválido.
07h: Coprocesador matemático no presente.

Ejemplo de ello es cuando se hace una división por cero, cuando la CPU encuentra un código de operación no válido, etc. En la **Tabla 2** hay un mapa de las excepciones que existen.
Las interrupciones hardware son aquellas que están reservadas para dispositivos físicos del sistema, como el teclado, unidades de disco, el reloj interno, etc, y que se ejecutan automáticamente cada vez que el dispositivo físico asociado (correspondiente) envía una señal de petición de interrupción *hard* (como ocurre, por ejemplo, cuando pulsamos una tecla). En la **Tabla 3** podemos ver una lista de las interrupciones hardware.
Por otro lado, tenemos que las interrupciones software son las que están disponibles para el programador y/o sistema operativo, y que sólo se ejecutan si las llamamos explícitamente mediante una instrucción que hay para tal fin y que hace las funciones de la instrucción *CALL* que, como deberíamos recordar de anteriores números, se usaba para llamar a procedimientos normales dentro de nuestro propio programa.

TABLA 3: MAPA DE INTERRUPCIONES HARDWARE

08h: Interrupción del temporizador.
09h: Teclado, se ejecuta con pulsación de tecla.
0Ah: Canal 1/0.
0Bh: Señal desde el COM2.
0Ch: Señal desde el COM1.
0Dh: señal desde LPT2, impresora.
0Eh: Señal enviada por controladora de diskette.
0Fh: Señal enviada por LPT1, impresora.
70h: Reloj de tiempo real.
71h: Desvío de la IRQ2.
72h: Reservada para el sistema.
73h: Reservada para el sistema.
74h: Reservada para el sistema.
75h: Señal procedente del Coprocesador (FPU).
76h: Señal de la Controladora de disco duro.
77h: Interrupción reservada para el sistema..

EL PRIMER GRUPO, INTERRUPCIONES BIOS

Las interrupciones BIOS son un total de 32 y comprenden desde la primera, la 00h hexadecimal, hasta la 1Fh, y se dividen en cinco subclases: las interrupciones internas del microprocesador (00h a 07h), las del controlador de interrupciones 8259 (08h a 0Fh), los puntos de entrada de la BIOS (10h hasta 1Ah), las rutinas del usuario (1Bh a 1Ch) y, por último, las que contienen parámetros de la BIOS (1Dh a 1Fh) y que no son realmente rutinas sino tan solo punteros a tablas de datos que usa la BIOS para tener información acerca del sistema sobre el cual está corriendo y otros relacionados.

INTERRUPCIONES DOS

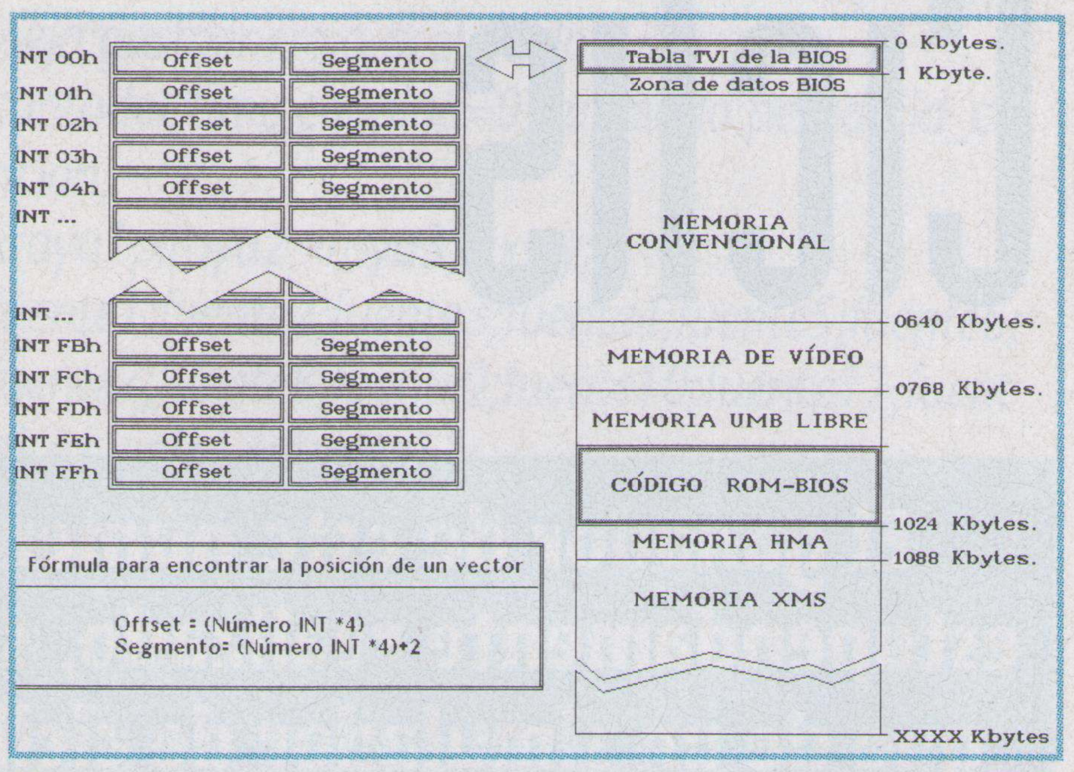
Las interrupciones del sistema Operativo DOS (MS-DOS se supone), son las que están contenidas entre la número 20h y la 3Fh (o sea, 32 interrupciones más), y algunas de las cuales poseen muchos servicios disponibles. La lista de las principales es ésta: la Int 20h, interrupción de terminación del programa, la 21h, que se usa para las peticiones de servicios, la 22h, que apunta a la dirección de terminación para cuando finalice el programa, la 23h, que posee la rutina encargada de ejecutar el proceso asociado a la pulsación de las teclas CTRL+ BREAK, la interrupción 24h, que es una rutina para el manejo de errores críticos, la 25h, que se usa para la lectura absoluta en disco, la 26h, que se usa para la escritura absoluta en disco, la 27h, que se usa para la creación de los famosos programas residentes y las restantes, las comprendidas entre las 28h y 3Fh, están reservadas para el sistema operativo y no son accesibles ni modificables.

INTERRUPCIONES DE USUARIO

Este grupo de interrupciones que son las comprendidas entre la 40h y la 7F h (un total de 64), están libres para poder ser usadas por el usuario (el programador).

COMO FUNCIONAN LAS INTERRUPCIONES

Como hemos dicho, para llamar a las interrupciones, que son rutinas ubicadas en memoria, sólo hace falta indicar su número, y no un puntero de memoria al inicio de la misma. ¿Cómo se entiende ello? pues bien, es muy fácil de entender.



ESQUEMA DE LA ESTRUCTURA Y UBICACION DE LA TVI EN MEMORIA.

En todos los sistemas hay, desde que se arranca el ordenador, un espacio de 4*256 bytes (1 kbyte) que empieza en la posición física 0 de memoria y que es una tabla con 256 punteros de memoria, donde cada 4 bytes corresponde a un valor de *OFFSET:SEGMENTO* (2+2 bytes, 2 palabras) con un puntero al inicio de una rutina o tabla de datos.
Cada número de interrupción corresponde a uno de estos punteros, y así cuando llamamos, por ejemplo, a la interrupción 21h del DOS con la instrucción **INT 21h**, lo único que hace la CPU es pasar el control de ejecución a la dirección indicada por el puntero contenido en la posición de memoria 4*21h de la tabla que hemos dicho y que empieza en el segmento 0000h (Dirección física 0), por lo que la dirección completa de la rutina se extrae de la posición de memoria: 00h:[4*21h].
A la tabla donde están todos los punteros de interrupción se la llama *Tabla de Vectores de Interrupción* (TVI), donde se considera el término vector un equivalente al de puntero. Para más detalles de cómo está diseñada esta tabla, pueden ver la fotografía número 1, donde hay un esquema de la misma.

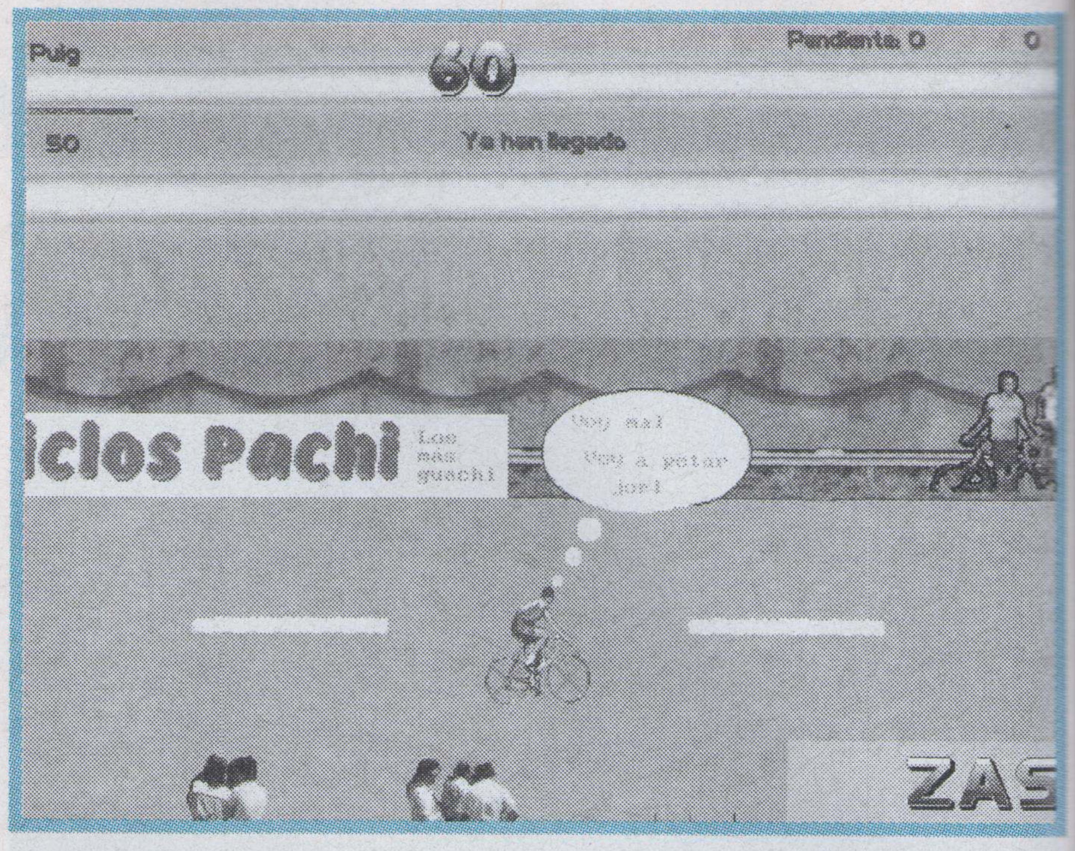
BIBLIOGRAFÍA

- Cómo programar en ensamblador 80x86. Biblioteca técnica de Programación. Prensa Técnica.
- 8088-8086/8087 Programación ENSAMBLADOR en entorno MS-DOS, Miguel Ángel Rodríguez Roselló. Anaya Multimedia.
- Programación del 80386/387. Manual de referencia y técnicas avanzadas de programación para diseñadores de sistemas, programadores y usuarios avanzados. John H. Crawford/Patrick P. Gelsinger. Anaya Multimedia.
- La ROM-BIOS de IBM. Ray Duncan. Anaya multimedia.
- Funciones del MS-DOS, Ray Duncan. Anaya Multimedia.

Autor: Luis Rodríguez Campos

JUEGOS GANADORES: 1º

Ciclis



El ganador de este número es un juego muy original y estupendamente realizado, se trata nada más y nada menos que de un simulador de ciclismo. Tendrás que dosificar las fuerzas de tu corredor para que llegue el primero a la meta y como fondo unos paisajes estupendos, casi como una retransmisión televisiva.



No conocemos muchos simuladores de ciclismo, de hecho éste es el primero que hemos visto, una idea muy buena y además se deja jugar muy bien. Lo que si sabemos es que una empresa española está ahora mismo realizando un simulador del mismo tipo. ¿Será tan bueno como *Ciclis*? El tiempo lo dirá. Enhorabuena Luis y sigue así. A este juego le falta muy poco para ser profesional, tan poco que nos ha dicho el propio autor que ya ha conseguido vender alguno a través de su página web. En el CD metemos la demo nada más, si te gusta ya sabes.

CARACTERISTICAS

- Gráficos fotorrealistas.
- Diversas músicas originales, sonidos y voces.
- Elevada inteligencia artificial.
- Numerosas competiciones reales en las que participar, desde la vuelta a Valencia a el Tour de Francia, pasando por la Paris-Niza.
- Completo editor del perfil de cada etapa, para que las construyas a tu gusto o copies las reales del actual Tour de Francia, por ejemplo.
- Manejo de los otros ciclistas de tu equipo por tí mismo o por otro jugador.
- Numerosos gráficos adaptados al tipo de carrera.
- Escala pendientes en las que 12 km/h es mucho y descensos en los que rozarás los 100 km/h.
- Comentarios referentes a lo que pasa en carrera dichos por los ciclistas.
- Los ciclistas se comportan como en una carrera real: se ponen a rueda, lanzan ataques cuando el pelotón va a poca velocidad, se quedan si vas muy rápido, sprintan en la meta, buscan la rueda buena...
- Puedes salvar tu partida, no se corre el Tour en un día...
- Completa información en carrera de la posición de los otros ciclistas.
- Modo "historia": empiezas en aficionados y tienes que mejorar tu fuerza y tu equipo para pasar a profesionales, y finalmente ganar una Gran vuelta por etapas.
- Puedes elegir corredor, desde Sodexo, Banaka, Olarra...hasta Banesto, Kelme y Mapei.

REQUERIMIENTOS

Se ha hecho con un Pentium 100,a si que puedes hacerte una idea, pero en modo de alta calidad y con 100 corredores, la cosa va un poco lentilla.

ACERCA DEL AUTOR...

En 1997 hice un *Ciclis* para Visual Basic (!) pero no llegué a hacer mas que una etapa. Luego hice un juego de estrategia que era entretenidillo, pero poco más. En verano de 1999 empecé el nuevo *Ciclis* con Div 2, un entorno de programación que facilita la inserción de gráficos, con lo que el *Ciclis* original fué retomado con fuerza. Acabado en Abril del 2000. Ahora estudio Informática de gestión en la Universidad de Córdoba.

CONTROLES

Arriba y abajo: evidentes.
Izquierda: frenar.
Derecha: Acelerar.
1,2...9: Canciones, aunque en la demo solo hay una.
0: Quitar la canción.
u: Quitar el ruido de los pedales.
F1: Disminuir volumen.
F2: Aumentar volumen.

Avanzar página: Mover la cámara a la derecha.
Fin: Volver con la cámara a tu corredor.
Suprimir: Mover la cámara a la izquierda.

C: Comer.
Alt-x: Salir del juego de golpe.
Escape: Salir de esta vuelta.
Pausa: Pausa.
Control-alt-p: Graba una captura del juego.

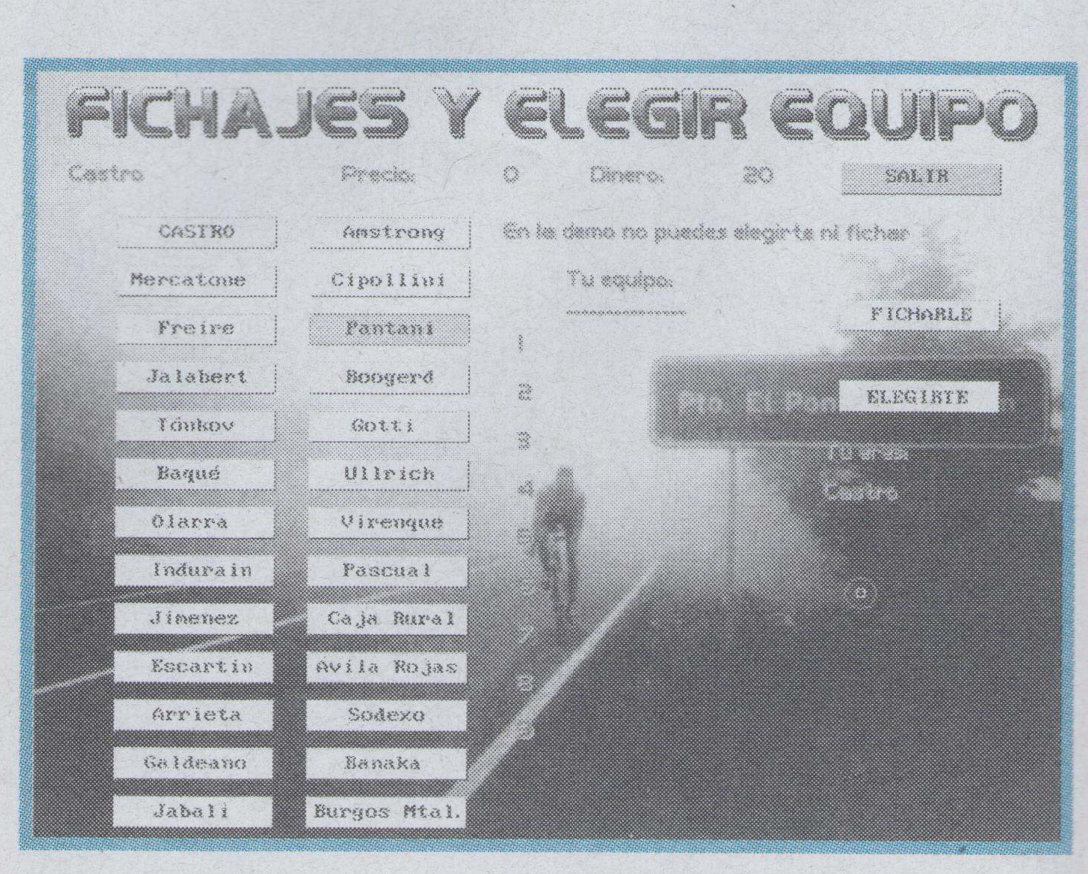
B: Aumentar brillo.
N: Disminuir brillo.
V: Rotar la intensidad de los colores primarios aleatoriamente.
Espacio: Brillo original.

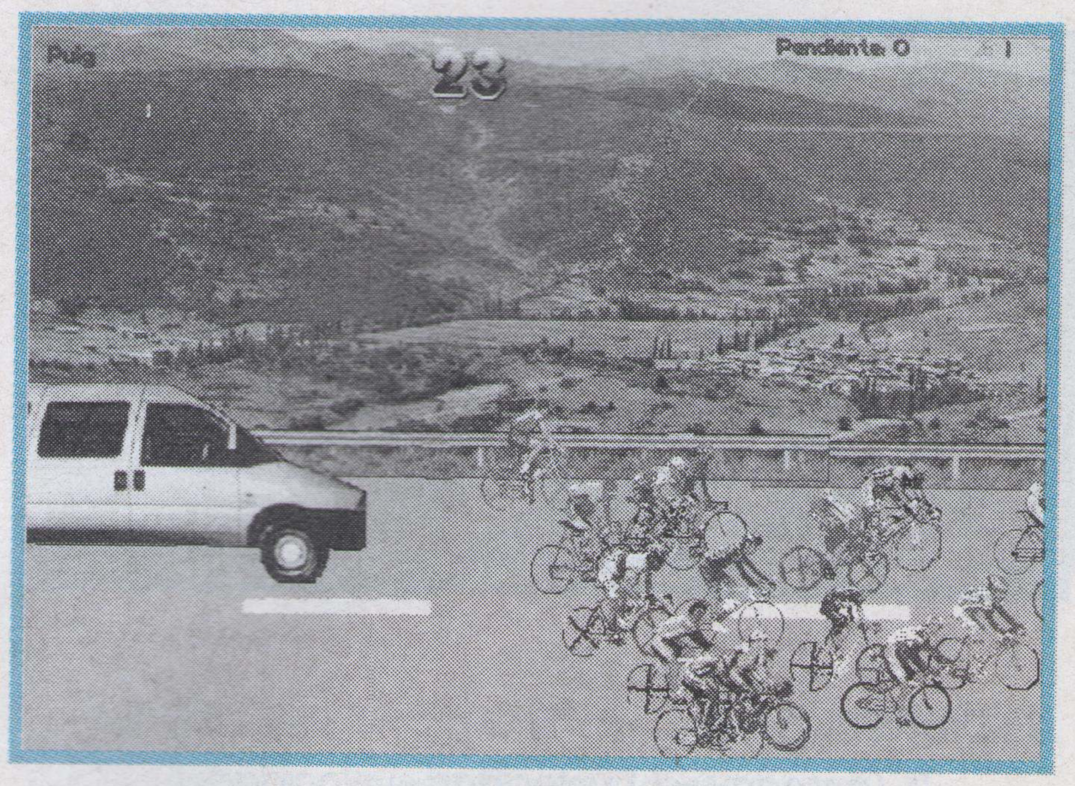
W, A, S, D: Para manejar al corredor de tu equipo que tengas seleccionado, pero como en la demo no tienes compañeros, nada. Con la versión entera se puede jugar "a dobles" si otro maneja al compañero.

Intro y Shift derecho: Cambia de compañero entre los que tengas.

INDICADORES DE PANTALLA

Arriba a la izquierda te dice el nombre de la vuelta. Justo debajo hay una barra con la energía, si llega a la raya que hay un poco más





adelante has petado y te frenas un poco. Debajo un número que indica lo que está variando la energía en estos momentos. En medio hay un número grande que indica la velocidad. A su izquierda sale la palabra 'CHUP' cuando estás chupando rueda, con lo que gastas menos energía.

Mas a la derecha te dicen la pendiente y un gráfico con el perfil de la etapa, con un punto rojo que simboliza tu posición. El numerito pequeño verde te dice las pantallas por segundo a los que te está rulando el juego, lo normal son 25. Puedes cambiarlo en 'opciones'. Aunque si tienes un ordenador del año 95 y le pones que vaya a 80, no llegará.

Arriba a la derecha viene el número que indica la comida que te queda. En la demo es solo 1 o ninguna (si ya has comido). Por último abajo del todo hay unas rayitas que representan la situación de los corredores en carrera: empiezan a la izquierda de la pantalla y la meta está al final,a la derecha. Tu eres la raya roja, los de tu equipo los verdes y el resto azules.

OPCIONES DEL JUEGO

En las pantallas del menú, cada vuelta indica con una lista de números el número de etapas que tiene esa vuelta y el tipo de etapa que es.

0: Etapa llana.
1: Media montaña.
2: Alta montaña.

En 'opciones' tipo de PC es para que los que les vaya lento desactiven algunas cosas:

0: No se mueve el fondo de la pantalla y los enemigos no chocan entre ellos.
1: Como el cero pero sí que chocan.
2: Todo activado.

Modo historia: Empiezas siendo un aficionado (sólo puedes correr las vueltas de aficionado, que están abajo en la lista de cada mes. Al darle a final de temporada, si has fichado gente y subido suficientemente tu energía, pasas al nivel 2,con lo que puedes correr con los profesionales, aunque todavía no lo seas. Las carreras de nivel 2 se reconocen porque están pintadas de morado. Si al acabar la temporada tienes más equipo y más energía, pasas a profesionales. Si no, se te mantiene acumulado para la siguiente temporada.

El editor de perfiles de etapa es bastante intuitivo: arriba incrementa la pendiente del tramo seleccionado, y abajo lo disminuye. Izquierda y derecha cambian de tramo.

Consejos
Procura chupar rueda siempre que puedas, controlando los ataques, decidiendo en que escapada te vás... Gastaté la comida (pulsar c) en el momento oportuno, cuando vayas a petar. Lanza el sprint en cuanto veas las vallas de meta.

Suerte,y gracias por jugar...... .

CÓDIGO FUENTE

```
//COMPILER_OPTIONS _MAX_PROCESS=200;
//MUCHO CUIDADO
//mirar con memory_free a ver si algun punto del
programa acumula gasto de memoria
//Haciendo tablas pueden simplificarse muchas
cosas,como con enemigo
PROGRAM Ciclis;          //.1999.Luis
Rodriguez Campos.Zasoft.
//a¤adir and v>=id2.v al compilar
GLOBAL a=1; elfondo1=312;
xrayametainicial=20000; etapa=0; //LAS 3
GRANDES
nivel_actual=3;dinero=0;comida=1;peta=25000;
pcbueno=2;
lame;lame2;lame3;lame4;lame5;lame6;lame7;la
me13;lame15;lame18;
music;c_por_equipo=1;nivel=1;tipo_etapa;
nvoltavieja;nvolta; volumen=256; s_pedal1;
dst_cabeza;dst_ultimo;
chupingvieja; vvieja; lamevieja; llamabocata;
llamabocata2;
salir=1;histori;
sonidoquit;paclasis;choca;vpeloton=27;en_carrer
a;esc_pulsed; borraya;
conversor[9];fichetotal;vobjetivo;quemadavieja;
xrayameta;popo;continuaractivo;string
vuelta="Leon";finished;
ponfoto;playergana;string etapaelegida;
tempoganador;tiempof;enx;eny;tp3;despierto;tem
p;
pr;distanciavieja;diego=49;dineroviejo;
fiched;fiched1;fiched2;fiched3;fiched4;
boton1;boton2;boton3;boton4;//mira a ver lo
que puedes poner private o que incluso sobre
maxim=50;
p[12];p1;p2;p3;p4;p5;p6;p7;p8;p9;p10;p11;p12
;vu1;vu2;vu3;vu4;vu5;vu6;vu7;vu41;
pendiente=0;pendientevieja;ingresos;
moral=10;distancia=0;velocidad=23;tiempo=0;y
vieja;gastoenergia;aceleracion=0;
contador;timo;chuping=0;quemada;petada;id_ca
ncion;
xraya1;xraya2;xraya3; xx;yy;
fuente1;fuente2;fuente3;fuente4;fuente5;fuente6;
fuente7;fuente8;fuente9;fuente10;fuente11;fuent
e12;fuente13;fuente14;
xfondo1;xfondo2;xfondo3;xvalla1;xvalla2;xvalla3
;xvalla4;xcamino1;xcamino2;
enemigor[101];fiche[25];
string conel1;string conel2;string conel3;string
conel4;string conel5;string conel6;string
conel7;string conel8;string conel9;
struct clasis    [101] distancias;tempo; string
nombre; end //borrar distancias?
struct clasisg   [101] tempofg;        string
nombre; end
struct clasistemp [101] temptotal;      string
nombre; end

 LOCAL
ex;ey;ficher;dst;v;dstvieja;
chupi1;chupi2;chupi3;chupi4;chupi5;chupi6;chup
i7;chupi13;chupi15;chupi18;
contador2;timo2;t;tempof;tmp;
vmin2;vmax2; pulsed;hacorrido;energy;string
name;
vmin;//velocidad minima a la que un enemigo
lanza un ataque
vmax;//     "    maxima        "
//redmin;//cuantos km/h reduce (como
minimo)cada 10s en un ataque
redmax;//        "           maximo      "
//tipmin;//cuanto tiempo tarda como minimo en
dar un ataque,cuando va a velocidad jose
tipmax;//        "       maximo       "

BEGIN
  set_fps(25,1);//me veo obligado a poner 1
  set_mode(m640x480);
  music=rand(1,3);
  if(music==1)
load_song("..\ciclis\ciclis1.xm",1);end
  if(music==2)
load_song("..\ciclis\ciclis2.xm",1);end
  if(music==3)
load_song("..\ciclis\ciclis3.xm",1);end
  song(id_cancion);
```



Juegos ganadores: 1º

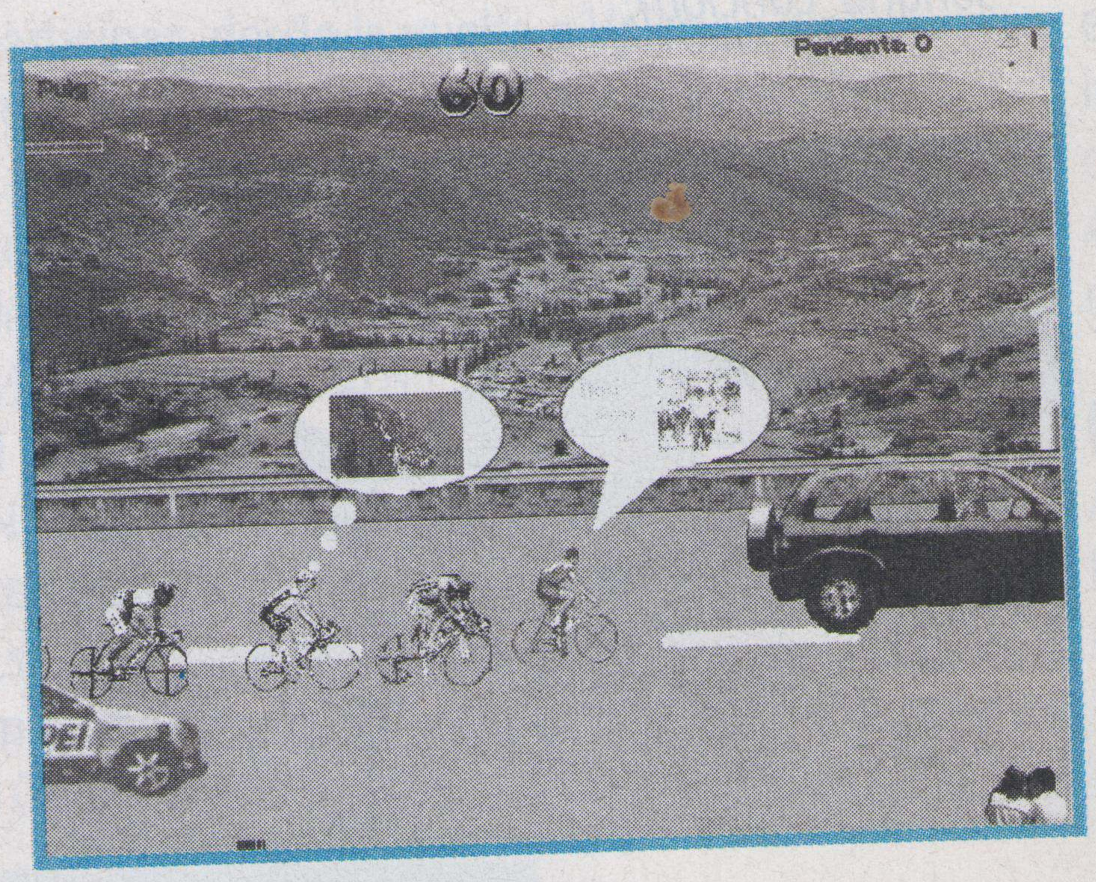
```
fuente1=load_fnt("..\ciclis\ciclis1.fnt");
fuente2=load_fnt("..\ciclis\ciclis2.fnt");
fuente3=load_fnt("..\ciclis\ciclis3.fnt");
fuente4=load_fnt("..\ciclis\ciclis4.fnt");
fuente5=load_fnt("..\ciclis\ciclis5.fnt");
fuente6=load_fnt("..\ciclis\ciclis6.fnt");
fuente7=load_fnt("..\ciclis\ciclis7.fnt");
fuente8=load_fnt("..\ciclis\ciclis8.fnt");
fuente9=load_fnt("..\ciclis\ciclis9.fnt");
fuente10=load_fnt("..\ciclis\ciclis10.fnt");
fuente11=load_fnt("..\ciclis\ciclis11.fnt");
fuente12=load_fnt("..\ciclis\ciclis12.fnt");
fuente13=load_fnt("..\ciclis\ciclis13.fnt");
fuente14=load_fnt("..\ciclis\ciclis14.fnt");
// dump_type=partial_dump;//quita esto si usas scrolls
// restore_type=partial_restore;

intro();
frame;
END
//————————————————————
//——————————PRINCIPAL————
//————————————————————

PROCESS principal()
PRIVATE rapid; id1;id2;id3; pop; esc; n=1; sprint; pe;
BEGIN id1=raya1();id2=raya2();id3=raya3();
graph=diego; y=300; x=320; xx=x;

s_pedal1=load_wav("..\ciclis\voces\pedal1.wav",1);
    LOOP //borraya=memory_free();
    if(pr==1)pr=0;end //la segunda vez que entre aqui
    IF (pr==2)pr=1;perfilin(); mouse.graph=327;
pendiente=0; angle=0; graph=diego; esc=0;

sonidoquit=sound(s_pedal1,volumen/4,256);player
gana=0; esc_pulsed=0;sprint=0;
    dst_cabeza=0;fade(90,90,90,100);
    for(n=1;n<=fichetotal;n++)
enemigor[conversor[n]].ficher=1; end
    if(historí==1 and nvoltavieja>=nvolta and
```




```
etapa==1 and salir==1)etapa=0; finished=0;
esc_pulsed=1; calculos(); end
  if(histori==1 and nivel!=nivel_actual)
etapa=0; finished=0; esc_pulsed=1; calculos();
end
  if(esc_pulsed!=1)/*nvoltavieja=nvolta;*/salir=0;
end //para que solo se asigne nvoltavieja a las que realmente se han corrido
  END
  if(key(_del))x=x+75;end if(key(_pgdn))x=x-75;end if(key(_end))x=320;end
  if (xrayameta<5000)x=320;end
  if (pcbueno>1)angle=pendiente*1000;end

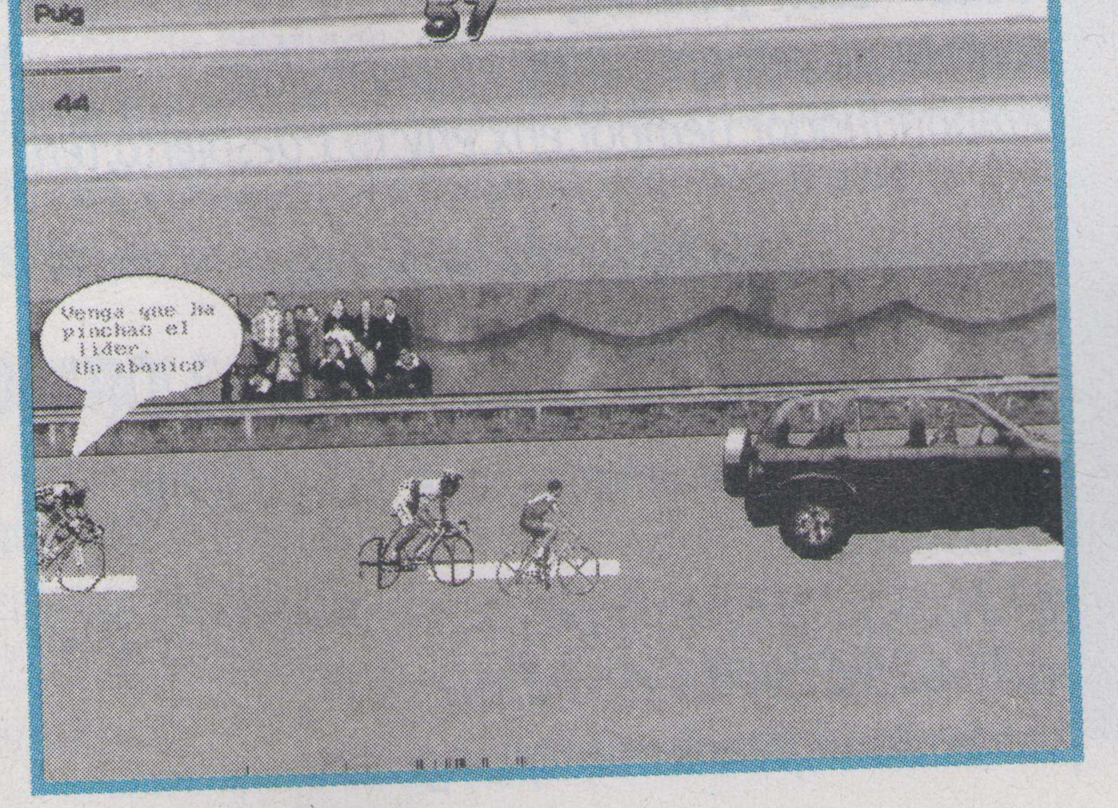
  if(pendiente>0) pe=3; else pe=6; end
  gastoenergia=velocidad-35+rapid+pe*pendiente-chuping;//O sea,en principio gatoenergia=velocidad,lo demas son a¤adidos.
  if (velocidad<10 and gastoenergia>-10)gastoenergia=-10;end//para que en pendientes del 14% no gaste energia hasta parado.

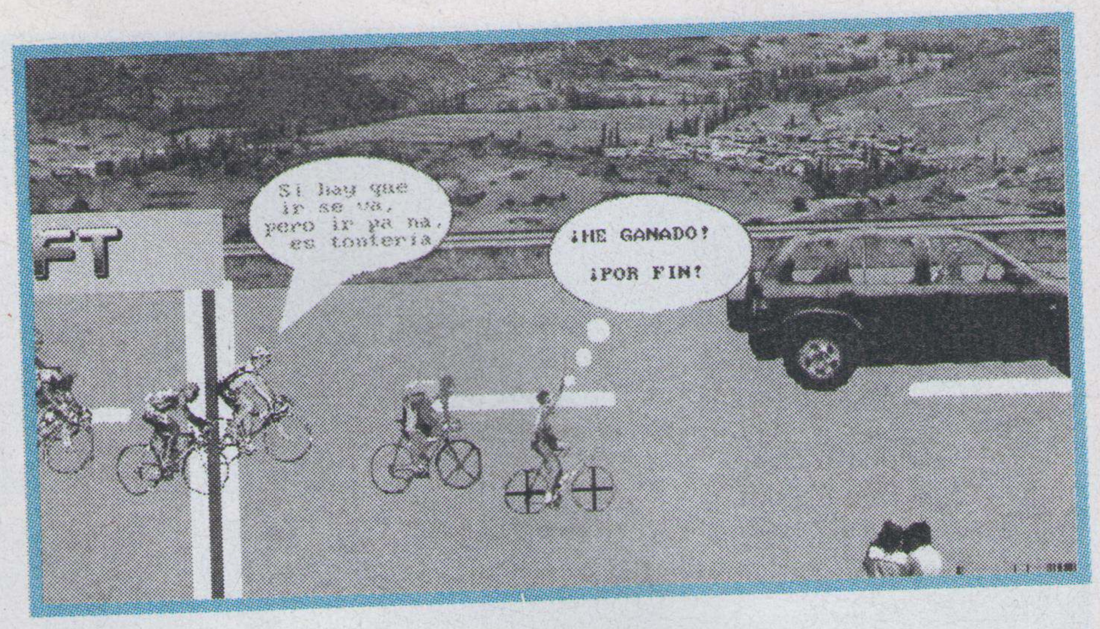
  if (velocidad>35) rapid=velocidad-35; else rapid=0; end//esto es que a partir de 35km/h gastas aun mas

  if (petada==0)
quemada=quemada+gastoenergia;end
//   if (quemada<quemadavieja) quemada-=10;
end //para que disminuya mas rapido,si no tarda mucho. Y no depende de la velocidad porque te recuperas lo mismo a 25 km/h que a 0 Km/h
  if (quemada<1) quemada=0; end
  if (petada==1)quemada=quemada-30;end
  if (quemada<1) quemada=0; end //lo repito para que no de problemas
//   varquemada=quemada-quemadavieja;
//   quemadavieja=quemada;

//————————————————Cambio de Velocidad————

/*
  if (key(_right))vobjetivo++;end if (key(_left))vobjetivo—;end
  IF (moral mod 3==0) //antes era 10,no 3,ver si
```



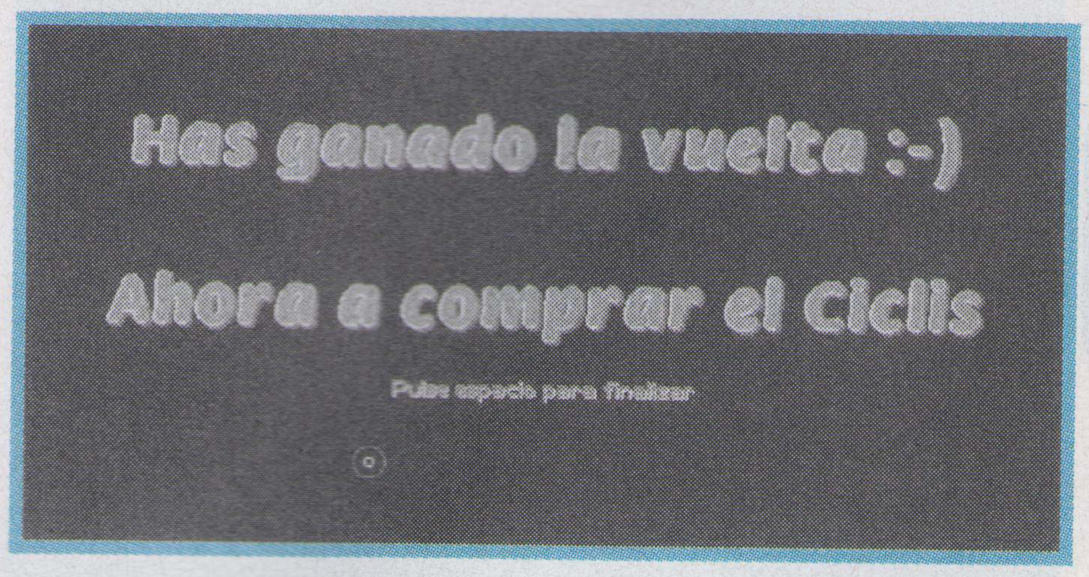



```
da problemas
    IF (petada==1) velocidad=velocidad-1 ;
    ELSE if (velocidad<vobjetivo)/*aceleracion=aceleracion+1;*/velocidad=velocidad+1;end
  END
  if (velocidad>vobjetivo)
velocidad=velocidad-1;end
    vobjetivo=velocidad;
        //DESHACERSE DE ACELERACION
//    if (key(_right)==0 and aceleracion>0)
aceleracion=aceleracion-5;end
//    if (aceleracion<0) aceleracion=0;end
  END
*/
  if(petada==1)velocidad—;end
  if(key(_right) and petada==0)velocidad++;
end
  if(key(_left)) velocidad—; end
//————————————————————

  if (velocidad<1) velocidad=0;end
  maxim=60-3*pendiente; //antes era maxim=50-2*pendiente;
  if (xrayameta<5000) sprint=1; end
  if (velocidad>maxim and xrayameta>5000)
velocidad=maxim;end
  if (velocidad>maxim+15)
velocidad=maxim+15; end
  if (pendiente==0 and velocidad>maxim)
velocidad=maxim; end

  //los cambios de personaje
  if (velocidad>0)contador=contador+2;
timo=200/velocidad;end
  IF (contador>timo and graph==diego-1)
graph=diego; contador=0; END
  IF (contador>timo and graph==diego)
graph=diego-1; contador=0; END
  IF (diego==49 or diego==46)
if (key(_right)    and graph>47)graph=46;
diego=46; end
  if (not key(_right)and graph<47)graph=49;
diego=49; end
  if (playergana==1) graph=47; end
  END

  z=-(y);
  //————————————————
PENDIENTE——————————
```

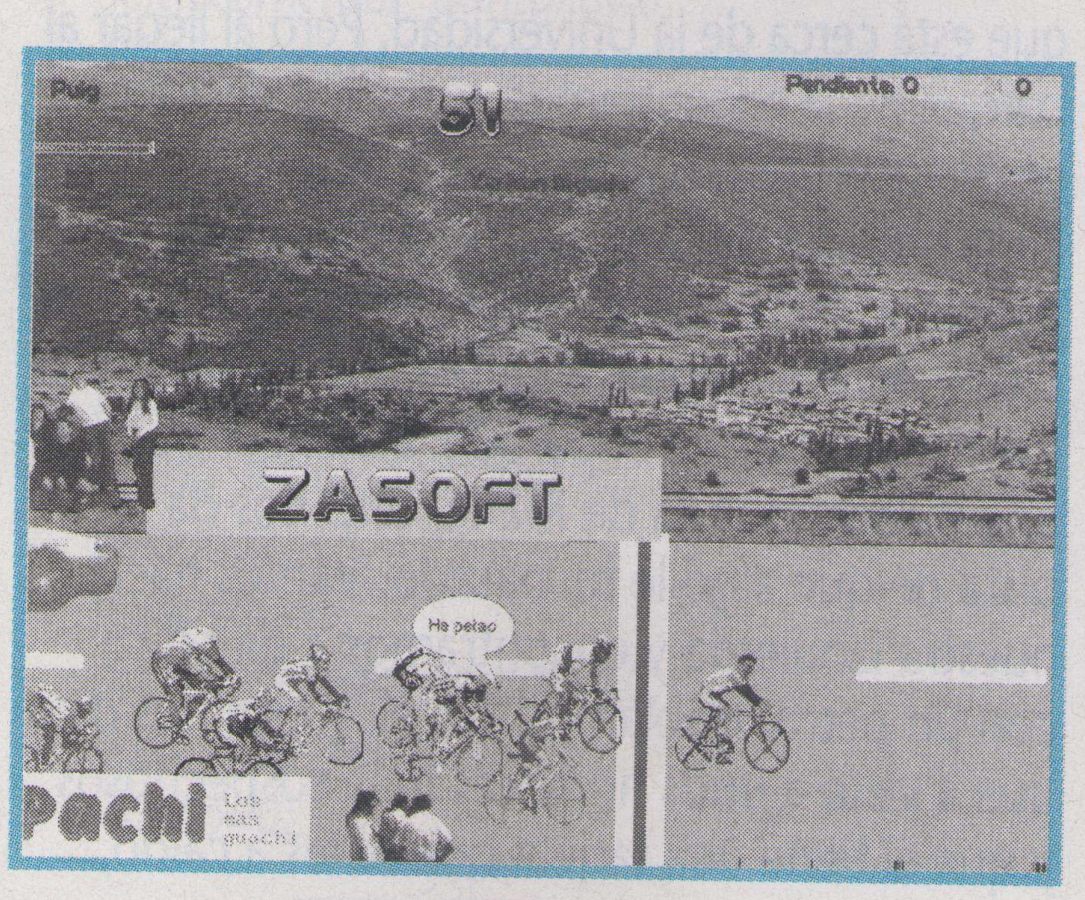



```
—
   pendientevieja=pendiente;
popo=xrayametainicial-xrayameta;
   //Lo de la pendiente es asi:

   IF (moral mod 25==0 and pcbueno<2)
pop=xrayametainicial/12;
      nvoltavieja=nvolta;
   IF (popo>0 and popo<pop       )
pendiente=p[1];end
   IF (popo>pop and popo<pop*2   )
pendiente=p[2];end
   IF (popo>pop*2 and popo<pop*3 )
pendiente=p[3];end
   IF (popo>pop*3 and popo<pop*4 )
pendiente=p[4];end
   IF (popo>pop*4 and popo<pop*5 )
pendiente=p[5];end
   IF (popo>pop*5 and popo<pop*6 )
pendiente=p[6];end
   IF (popo>pop*6 and popo<pop*7 )
pendiente=p[7];end
   IF (popo>pop*7 and popo<pop*8 )
pendiente=p[8];end
   IF (popo>pop*8 and popo<pop*9 )
pendiente=p[9];end
   IF (popo>pop*9 and popo<pop*10 )
pendiente=p[10];end
   IF (popo>pop*10 and popo<pop*11)
pendiente=p[11];end
   IF (popo>pop*11 and popo<pop*12)
pendiente=p[12];end
   END
   IF (moral mod 25==0 and pcbueno==2)
pop=xrayametainicial/12;
      nvoltavieja=nvolta;
   IF (popo>0 and popo<pop        and
pendiente!=p[1])if(pendiente>p[1])pendiente—;
else pendiente++;end END
   IF (popo>pop and popo<pop*2    and
pendiente!=p[2])if(pendiente>p[2])pendiente—;
else pendiente++;end END
   IF (popo>pop*2 and popo<pop*3   and
pendiente!=p[3])if(pendiente>p[3])pendiente—;
else pendiente++;end END
   IF (popo>pop*3 and popo<pop*4   and
pendiente!=p[4])if(pendiente>p[4])pendiente—;
else pendiente++;end END
   IF (popo>pop*4 and popo<pop*5   and
pendiente!=p[5])if(pendiente>p[5])pendiente—;
else pendiente++;end END
   IF (popo>pop*5 and popo<pop*6   and
pendiente!=p[6])if(pendiente>p[6])pendiente—;
else pendiente++;end END
   IF (popo>pop*6 and popo<pop*7   and
pendiente!=p[7])if(pendiente>p[7])pendiente—;
else pendiente++;end END
   IF (popo>pop*7 and popo<pop*8   and
pendiente!=p[8])if(pendiente>p[8])pendiente—;
else pendiente++;end END
   IF (popo>pop*8 and popo<pop*9   and
pendiente!=p[9])if(pendiente>p[9])pendiente—;
else pendiente++;end END
   IF (popo>pop*9 and popo<pop*10  and
pendiente!=p[10])if(pendiente>p[10])pendiente
—; else pendiente++;end END
   IF (popo>pop*10 and popo<pop*11 and
pendiente!=p[11])if(pendiente>p[11])pendiente
—; else pendiente++;end END
   IF (popo>pop*11 and popo<pop*12 and
pendiente!=p[12])if(pendiente>p[12])pendiente
—; else pendiente++;end END
   END
   if (pendiente!=0 and pcbueno>1)
signal(id1,s_sleep);signal(id2,s_sleep);signal(id3,s
_sleep);
   else
signal(id1,s_wakeup);signal(id2,s_wakeup);signal
(id3,s_wakeup);end
//   if (key(_l)) pendiente—;end if (key(_o))
pendiente++;end
//————————————————————————————————
————————————————————————————————
   tiempo=1+timer/100;

   distanciavieja=distancia;//esto ha de ir antes
que la linea siguiente para que funcione chupi1
   distancia=distancia+10*velocidad;
   moral=moral+1;//la moral solo es un contador
que suma 25 por segundo

   //ahora los controles de parriba y pabajo
   IF (NOT fading AND
key(_v))fade(rand(70,130), rand(70,130),
rand(70,130),1);END
   if (key(_space)) fade(90,90,90,1); end
   if (key(_b)) fade(110,110,110,1); end
   if (key(_n)) fade(60,60,60,1); end
```




```
//   if (key(_esc)) esc=tiempo; write
(fuente4,320,100,4,"PRESIONE ENTER PARA
ABANDONAR"); end
   if (key(_esc)/* and tiempo<esc+3*/)
en_carrera=0;  etapa=0;  finished=0;
esc_pulsed=1; stop_sound(sonidoquit); calculos();
end

   if (yy!=y) y=yy; end
   choca = collision(TYPE enemigo); //puede que
de problemas si colisiona al empezar,antes de
que haya cargado los proceso enemigos
   IF (choca<>0)
//   if (choca.v<velocidad-20 and tiempo>10)
//caidas
//
chupi1=0;chupi2=0;chupi3=0;chupi4=0;chupi5=
0;chupi6=0;chupi7=0;chupi13=0;chupi15=0;chu
pi18=0;
//     velocidad=0; choca.v=1;
//   end
   if(y*25<choca.y+500 and y*25>choca.y-
1)y=y+4;end
   if(y*25>choca.y-500 and y*25<choca.y)y=y-
4;end //copiar este concepto al player
   END
//   y=yy;
   if (key(_up)) y=y-4; end    if (y<265)
y=265;end
   if (key(_down)) y=y+4; end   if (y>440)
y=440;end
   yy=y;xx=x;

//——————————————llamadas a
bocatas——————————————
   if(velocidad<vvieja and chuping==0 and
chupingvieja>0) llamabocata=411; end
   if(lame<lamevieja and lame<3 and
velocidad>maxim-5)llamabocata=406;end
   chupingvieja=chuping; vvieja=velocidad;
lamevieja=lame;

lame=0;lame2=0;lame3=0;lame4=0;lame5=0;la
me6=0;lame7=0;lame13=0;lame15=0;lame18=
0;
   chuping=0;//no se puede poner een el proceso
enemigos pues lo leen muchos y entonces alguno
se lo asigna siempre


   if
(distancia>dst_cabeza)dst_cabeza=distancia;end
   if
(distancia<dst_ultimo)dst_ultimo=distancia;end
   dst_ultimo=1000000;

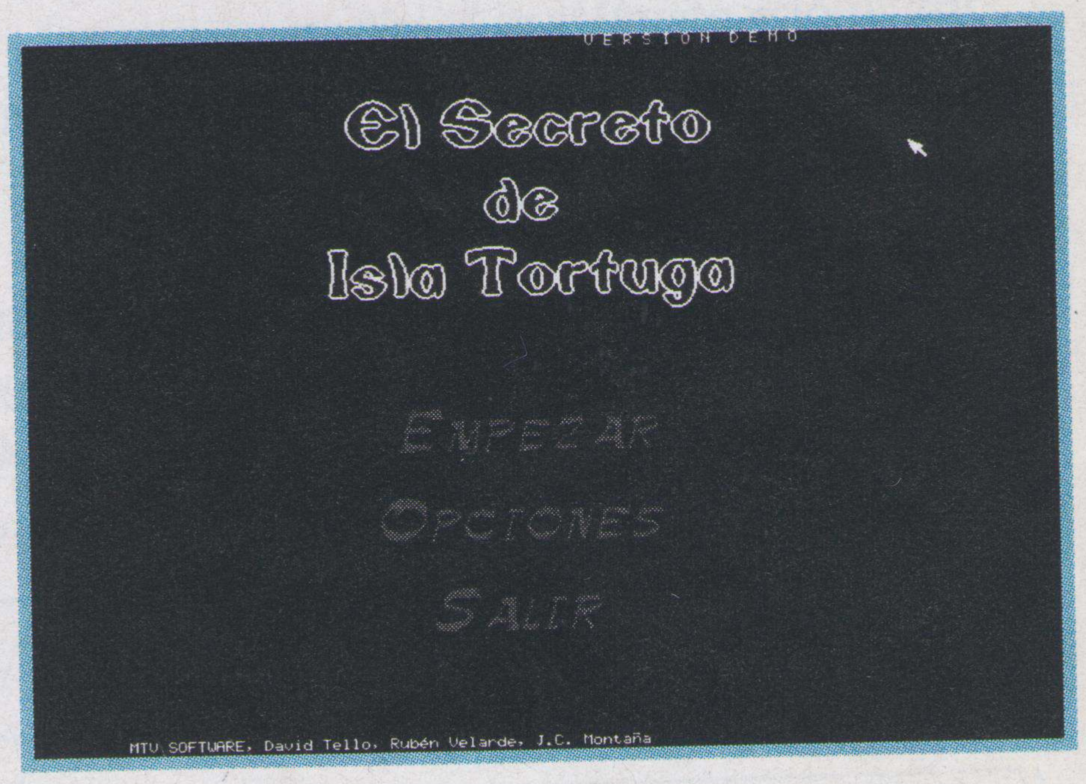
FRAME; END END

//————————————————————————————————
```

Autores: Juan Carlos Montaña
David Tello y Rubén Velarde

El Secreto de Isla Tortuga

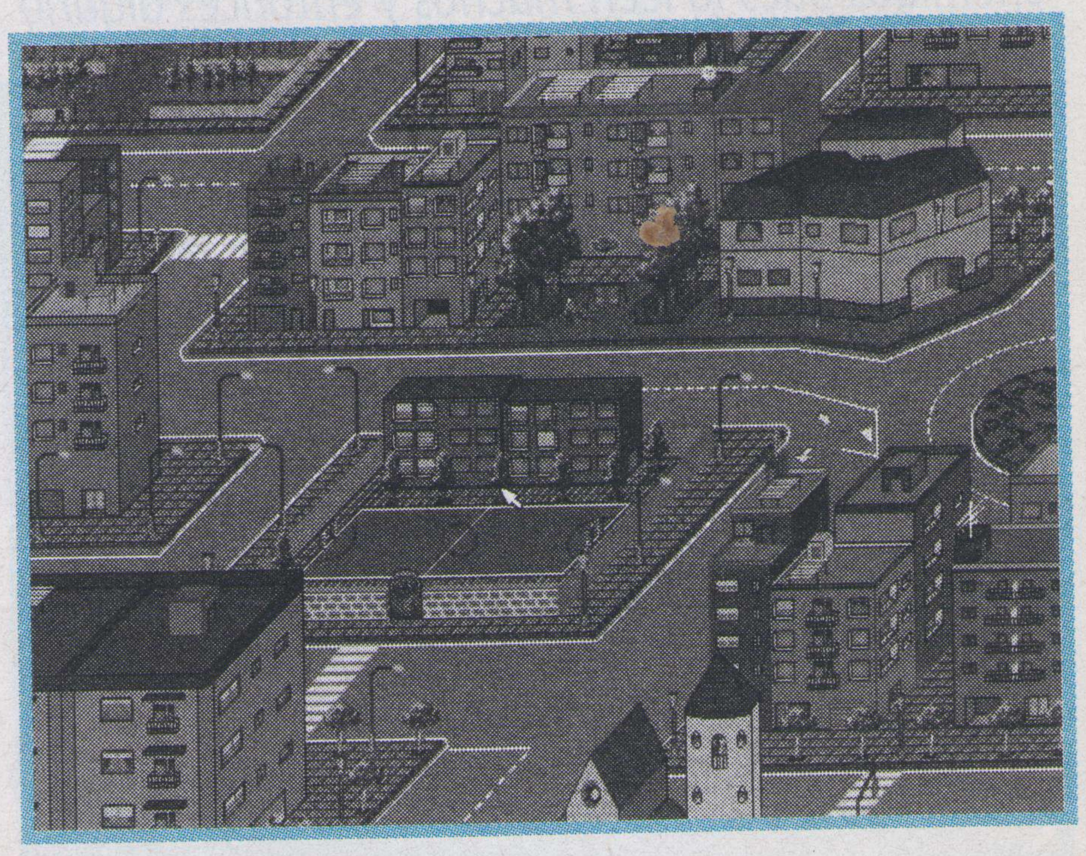
El segundo premio de nuestro concurso es una aventura gráfica en el que tendremos que estrujarnos un poco el cerebro para encontrar la solución de los problemas que se nos irán planteando, aunque los autores nos han hecho llegar una guía para aquellos que se atoren más de la cuenta.



Como dicen los autores de El secreto de la Isla de la Tortuga, no es muy usual ver aventuras gráficas hechas en DIV, aunque nosotros recordamos una en la que el protagonista tenía que meterse en un estadio de fútbol. Éste debe ser el segundo o tercer juego de este género que nos ha llegado a la redacción y merece un segundo premio, aunque esté incompleta. Quizás su código anime a otros a realizar este tipo de juegos. Nuestra enhorabuena a los ganadores.

LA OPINION DE LOS PROGRAMADORES

Nuestra opinión sobre el juego es que no es un juego muy corriente en Div, además no sabíamos nada de programar, así que nos lanzamos a hacer la aventura, que como se puede comprobar es un lío de comandos y demás, es casi imposible incluso para nosotros saber como lo hicimos.



DE DONDE PARTIO LA IDEA?

La idea partió de la base que veíamos que la gente solo hacia juegos de coches, lucha, y lo que más, de arcade. Empezamos pues a intentar ver si era posible hacer que el ratón detectase objetos, y que los pudiéramos coger, y una vez que lo conseguimos nos metimos en el argumento de la historia, que no está muy elaborado pero es válido.

DIFICULTADES

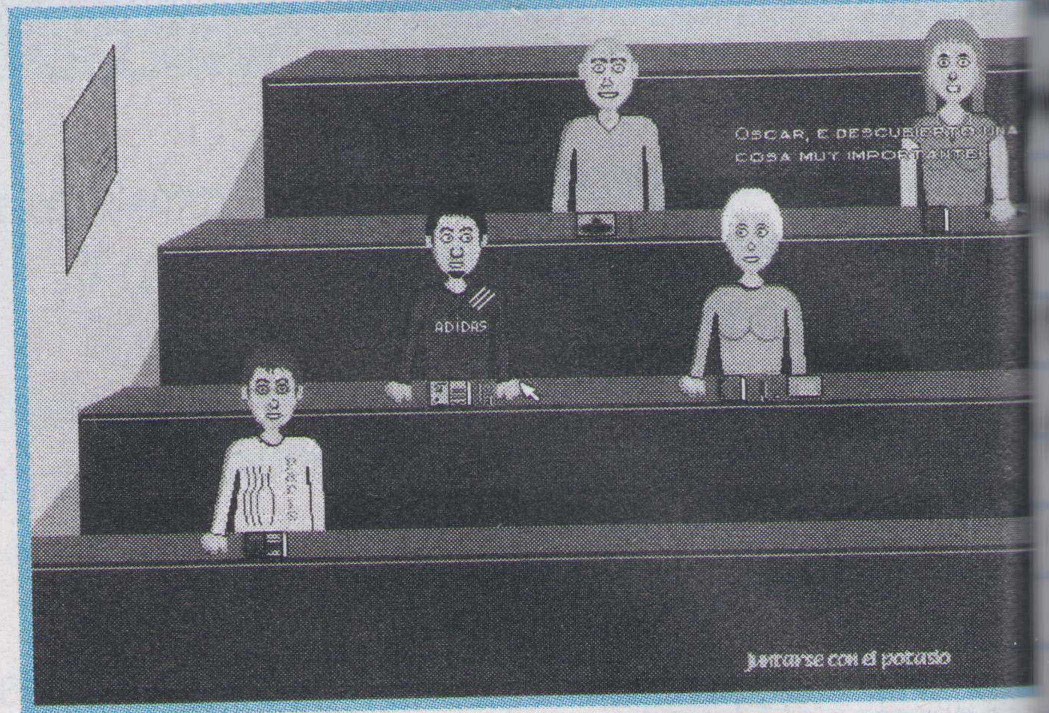
Las mayores dificultades fueron los dibujos, ya que entre nosotros tres solo hay dos que dibujan lo suficientemente bien, y entre los dos, cada uno tenía una forma diferente, por lo que nos tuvimos que adecuar al juego. Muchos dibujos los retocamos y los retocamos porque no había forma que quedaran bien, y aún así hay muchos fallos.

¿QUÉ ES LO MEJOR DE LA AVENTURA?

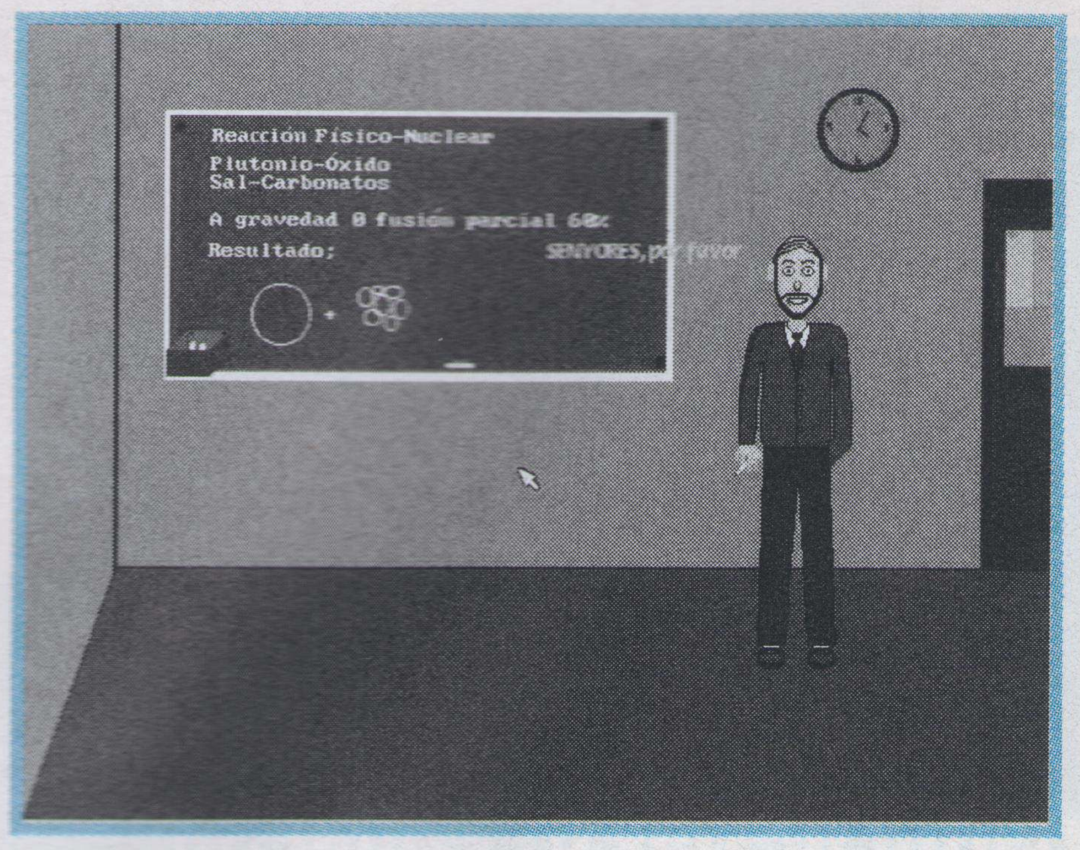
Para nosotros lo mejor es el mapa, porque fue muy difícil crearlo, debido a que para empezar no sabíamos nada de Div, nos guiábamos por lo que otras personas hacían, y no nos servía, porque no lo entendíamos, así que tuvimos que ir probando, hasta que un día acertamos, y muy bien (para nosotros).

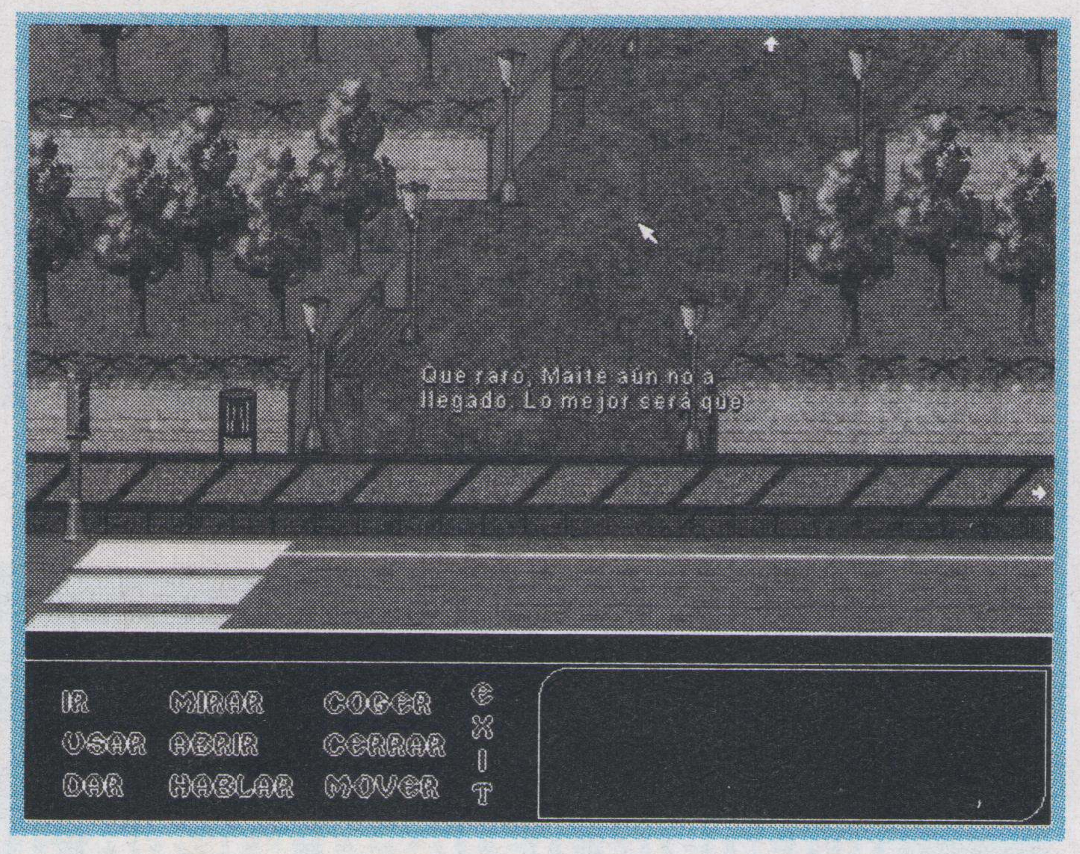
SOLUCIÓN DEL JUEGO

Nuestra amiga Maíte ha descubierto una cosa muy importante, por eso nos cita en el parque que está cerca de la Universidad. Pero al llegar al parque no la encontramos, y por eso decidimos ver si encontramos algún rastro de ella.
En la entrada del parque, podemos ver una papelera, en la cuál al mirar vemos que hay un chicle dentro. Seguimos adelante, y vamos a la fuente que hay en el parque, allí encontramos una pulsera que es de Maíte, esto nos da la pista de que algo le ha pasado. Vamos hacia nuestra izquierda y encontramos a un chico con un perro. Hablamos con él, y este nos dice que a visto a unos hombres muy sospechosos, y que el tiene una tarjeta que se les a caído,



nuestro objetivo es conseguir esa tarjeta, pero el no nos la quiere dar, hasta que después de insistir bastante, nos dice que nos la dará si le conseguimos una pelota que se le ha colado en un árbol. Así pues, solo hemos de buscar el árbol, que está a nuestra izquierda, pero al intentar coger la pelota no llegamos, necesitamos algo para alcanzarla.
Vamos hasta la fuente de nuevo, y ahora cogemos el camino de la derecha, en primer lugar encontramos a un jardinero, que está regando como cada día, y seguimos hacia la derecha hasta encontrar una caseta, abrimos la puerta, y entramos.
Dentro de ésta encontramos una escalera, la cuál cogemos, y volvemos al sitio donde estaba el árbol, usamos la escalera con el árbol y cogemos por fin la pelota. Ahora se la damos a el chico, y este nos da la ansiada tarjeta.
Ahora ya podemos irnos del parque. En el mapa, tenemos 3 sitios donde pulsar, uno es el parque otra vez, el otro la Universidad, y el último nuestra casa. Podemos ir a nuestra casa y allí subimos hasta nuestro piso. Intentamos abrir la puerta, pero no podemos porque está cerrada con llave, por eso miramos debajo de la esterilla, allí encontramos la llave, la usamos con la puerta y entramos, vamos por la casa, y no encontramos nada que nos pueda servir.
Nos vamos a la calle otra vez, y vamos a la Universidad. Allí podemos ir a ver al director, y nos interesa bastante entrar en su despacho, para eso tenemos que ir al lavabo de chicos, y allí usar el chicle con la pica, pero éste no se queda enganchado, así que tendremos que volver al parque y usar el chicle en la resina que suelta el árbol. Una vez lo hayamos hecho, usamos otra vez el chicle con la pica, y accionamos el grifo. Lo siguiente será una inundación enorme, por eso el director saldrá de su despacho y por fin podremos entrar. Y aquí se acaba esta pequeña demo de ejemplo.





CODIGO FUENTE

```
program Esdit;
global
    fichero,fichero2,fichero3;
fuente,fuente2,fuente3,fuente4,fuente5,fuente6,f
uente7,fuente8,fuente9,fuente10;
    creditos,credito2;
    titulo,titulo2,titulo3,titulo4,menu1,menu2;
    opcion=0;
    pantalla=0;
    usar_objeto;
    dar_objeto;
    hablar_con_persona=1;
    x1,y1;
    puerta;
    felpudo;
    puerta_abierta_casa;
    usar=0;
    numero=0; //sirve para el texto de las
personas
    conversacion;
    pide_balon=0;
    boca_persona1;
    boca_persona2;
    boca_director;
    video_inundacio=0;
    primera_vez_despacho=0;
    stop_saludo=0;
    activa_ojos=0;
    pa=2;
    acceso_denegado_pasillo_inundado=0;
    despacho_con_luz=0;
    acess=0;
    credits;
    mapa=0; //detecta_caminos con
start_scroll(0,...)

    //-objetos juego- que se pueden coger
    chicle=2;
    pulsera=0;
    tarjeta=0;
    escalera=0;
    pelota=0;
    llave=2;
    chicle_pegajoso=0;
    no_chicle_pegajoso=0;
    desague_taponado=0;
    grifo_encendido=0;
    //——-

    i=0; //variable de tiempo (contador)

    //objetos para inventario
    chicle_inventario;
    pulsera_inventario;
    escalera_inventario;
    pelota_inventario;
    tarjeta_inventario;
    llave_inventario;

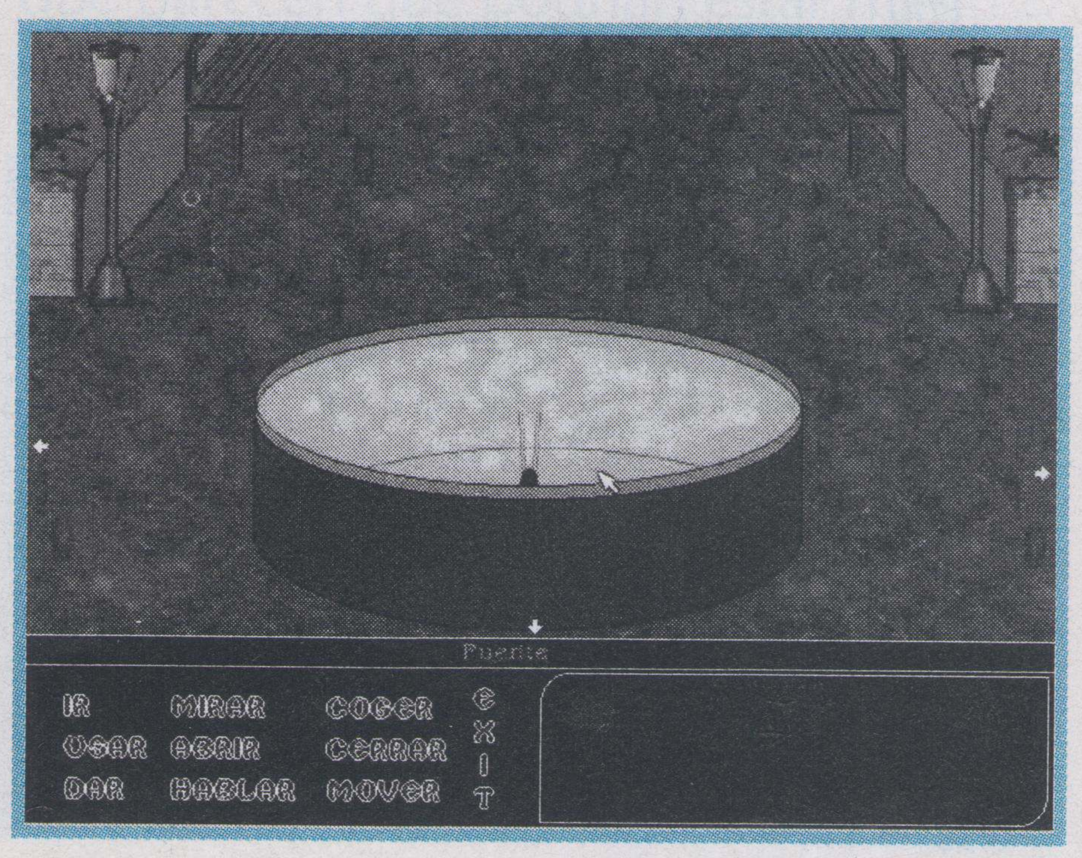
    //——————
    papelera_pantalla=0;
    chicle_pantalla=0;
    pulsera_pantalla=0;
    puerta_pantalla=0;
    escalera_pantalla=0;
    cortacesped_pantalla=0;
    cubo_pantalla=0;
    pelota_pantalla=0;
    arbol_pantalla=0;
    perro_pantalla=0;
    font_pantalla=0;
    lata_pantalla=0;
    saco_pantalla=0;
    escoba_pantalla=0;
    biblioteca_pantalla=0;
    biblioteca2_pantalla=0;
    felput_pantalla=0;
    llave_pantalla=0;
    puerta_casa_pantalla=0;
    puerta_habitacion_pantalla=0;
    sofa_comerdor_pantalla=0;
    televisor_pantalla=0;
    boton_pantalla=0;
    puertalavhom_pantalla=0;
    lavhomdentro_pantalla=0;
    grifo_lavabo_pantalla=0;
    pica_lavabo_pantalla=0;
    resina_arbol_pantalla;
    persona1=0;
    persona2=0;
    puerta_despac_pantalla=0;
    puer_desp_pantalla=0;
    videoo=0;
    viven=0;
    empezar;
    ha=0;
    ton=0;

begin
    set_mode(m640x480);
    fichero=load_fpg("e:\esdit\fpg\esdit.fpg");
    fichero3=load_fpg("e:\esdit\fpg\movis.fpg");
    fuente=load_fnt("e:\esdit\fnt\accion.fnt");
    fuente2=load_fnt("e:\esdit\fnt\accion2.fnt");
    fuente3=load_fnt("e:\esdit\fnt\barra.fnt");
    fuente4=load_fnt("e:\esdit\fnt\habla.fnt");
    fuente5=load_fnt("e:\esdit\fnt\habla2.fnt");
    fuente6=load_fnt("e:\esdit\fnt\texto.fnt");
    fuente7=load_fnt("e:\esdit\fnt\texto2.fnt");
    fuente8=load_fnt("e:\esdit\fnt\habla4.fnt");
    fuente9=load_fnt("e:\esdit\fnt\habla5.fnt");
    fuente10=load_fnt("e:\esdit\fnt\habla7.fnt");
    titulo=load_fnt("e:\esdit\fnt\titulo.fnt");
    titulo2=load_fnt("e:\esdit\fnt\titulo2.fnt");
    titulo3=load_fnt("e:\esdit\fnt\continua.fnt");
    titulo4=load_fnt("e:\esdit\fnt\titulo3.fnt");
    menu1=load_fnt("e:\esdit\fnt\menu.fnt");
    menu2=load_fnt("e:\esdit\fnt\menu2.fnt");
    creditos=load_fnt("e:\esdit\fnt\creditos.fnt");
    credito2=load_fnt("e:\esdit\fnt\credito2.fnt");
    fade_on();

    if(hablar_con_persona==0 or
hablar_con_persona==1)
        raton();
    end
    menu();

    loop
        delete_text(all_text);
        clear_screen();

        poner_fondo_pantalla();
        if(videoo==1)
         if (hablar_con_persona==0)
            saber_donde_se_pulsa();
            objetos_inventario();
            detecta_objetos_inventario();
detectarse_objetos_inventario_entre_si();
            poner_letras_accion();
```



Juegos ganadores 2º




```
            if (pantalla>0)
                put(fichero,202,320,240);
            end
        end
    end
    personajes();
    frame;
  end
end


process objeto(graph,size,x,y);
begin
    xput(fichero,graph,x,y,0,size,0,0);
end


process poner_letras_accion();
begin
    if (mouse.x>31 and mouse.x<49 and
mouse.y>407
        and mouse.y<420)
        write(fuente2,32,410,4,"IR");
        if (mouse.left)
            opcion=1;
        end
    else
        write(fuente,32,410,4,"IR");
    end

    if (mouse.x>98 and mouse.x<157 and
mouse.y>407
        and mouse.y<420)
        write(fuente2,120,410,4,"MIRAR");
        if (mouse.left)
            opcion=2;
        end
    else
        write(fuente,120,410,4,"MIRAR");
    end

    if (mouse.x>194 and mouse.x<260 and
mouse.y>407
        and mouse.y<420)
        write(fuente2,220,410,4,"COGER");
        if (mouse.left)
            opcion=3;
        end
    else
        write(fuente,220,410,4,"COGER");
    end

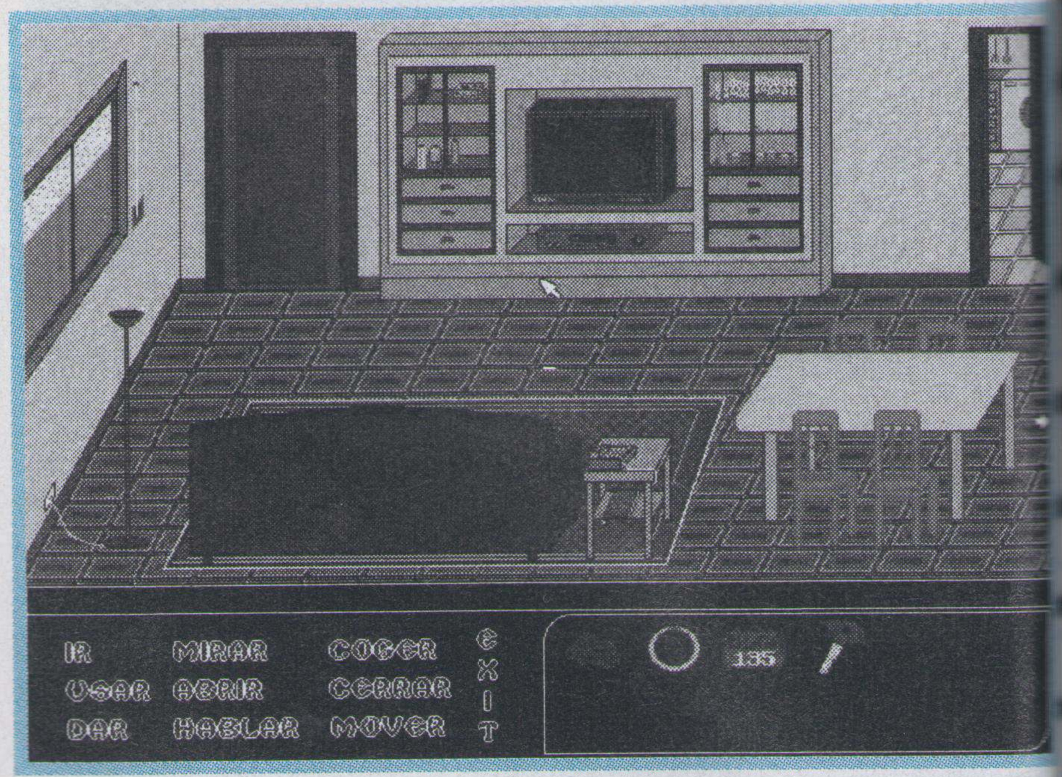
    if (mouse.x>31 and mouse.x<84 and
mouse.y>432
        and mouse.y<446)
        write(fuente2,50,435,4,"USAR");
        if (mouse.left)
            if(usar_objeto>0 or dar_objeto>0)
            usar_objeto=0;
            dar_objeto=0;
            end
            opcion=4;
        end
    else
        write(fuente,50,435,4,"USAR");
    end

    if (mouse.x>98 and mouse.x<154 and
mouse.y>432
        and mouse.y<446)
        write(fuente2,119,435,4,"ABRIR");
        if (mouse.left)
            opcion=5;
        end
    else
        write(fuente,119,435,4,"ABRIR");
    end

    if (mouse.x>194 and mouse.x<270 and
mouse.y>432
        and mouse.y<446)
        write(fuente2,225,435,4,"CERRAR");
        if (mouse.left)
            opcion=6;
        end
    else
        write(fuente,225,435,4,"CERRAR");
    end

    if (mouse.x>31 and mouse.x<70 and
mouse.y>457
        and mouse.y<470)
        write(fuente2,44,460,4,"DAR");
        if (mouse.left)
            if(usar_objeto>0 or dar_objeto>0)
            usar_objeto=0;
            dar_objeto=0;
            end
            opcion=7;
        end
    else
        write(fuente,44,460,4,"DAR");
    end

    if (mouse.x>98 and mouse.x<176 and
mouse.y>457
        and mouse.y<470)
        write(fuente2,130,460,4,"HABLAR");
        if (mouse.left)
            opcion=8;
```




```
        end
    else
        write(fuente,130,460,4,"HABLAR");
    end

    if (mouse.x>194 and mouse.x<265 and
mouse.y>457
        and mouse.y<470)
        write(fuente2,223,460,4,"MOVER");
        if (mouse.left)
            opcion=9;
        end
    else
        write(fuente,223,460,4,"MOVER");
    end

    if (mouse.x>283 and mouse.x<301 and
mouse.y>400
        and mouse.y<475)
        write(fuente2,285,404,4,"E");
        write(fuente2,285,424,4,"X");
        write(fuente2,285,444,4,"I");
        write(fuente2,285,464,4,"T");
        if (mouse.left)
            ha=0;
            clear_screen();
            hablar_con_persona=1;
            stop_scroll(0);
            pantalla=0;
            videoo=0;
            empezar=0;
            viven=2;
            graph=0;
            menu();
        end
    else
        write(fuente,285,404,4,"E");
        write(fuente,285,424,4,"X");
        write(fuente,285,444,4,"I");
        write(fuente,285,464,4,"T");
    end
end


process detecta_objetos();
private
    //se escriben aqu¡ los objetos porque no se
pueden coger
    papelera=0;
```

Autor: V. Reyes, C. López, M. López

Yamis Arena

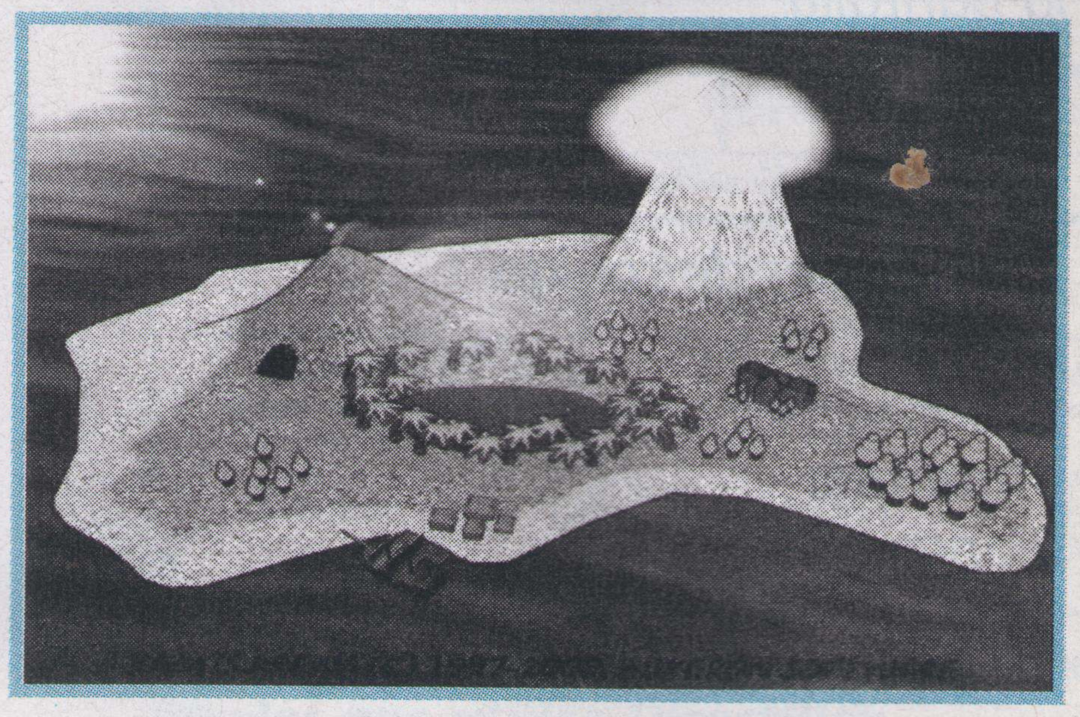
Este grupo de programadores ya se llevaron el primer premio del concurso en el número cuatro de la revista con Mr. Bones (o vete al Infierno). Ahora repiten mención honorífica con Yamis Arena un juego que se anuncia como el primero programado en Div que admite juego en red.

Nosotros en la redacción no hemos conseguido hacerle funcionar en multijugador, si lo hubiésemos hecho se hubiese llevado el primer premio, sin duda. Los autores aseguran que lo han probado en varios ordenadores y funciona, puede que sea problema de nuestros equipos. De cualquier modo, reciben el tercer premio, y recomendamos que estudiéis bien el código del juego para aprovechar su hallazgo si tenéis en mente programar algún juego con opción multijugador.

HOLA DE NUEVO

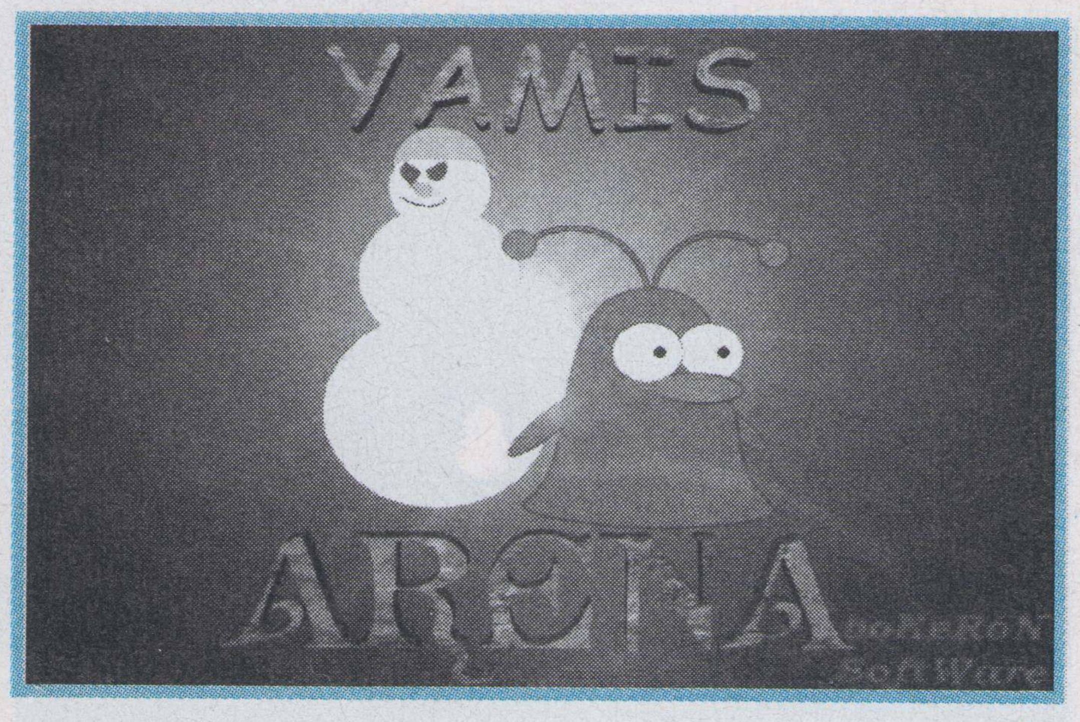
Aquí estamos de nuevo, con otro juego listo que además ¡¡funciona en red!! Ya sabemos que hay mucha gente que afirma que no funcionan, pero nosotros hemos conseguido hacerlo funcionar (de pura suerte, la verdad).

No fue nada fácil averiguar cómo se programaban juegos en red, pero finalmente lo logramos. El problema es que todo el mundo ve al servidor, pero los demás jugadores no se ven entre sí, y el servidor no los ve. La verdad probamos de todo: usar sólo el campo cero de la estructura, como el servidor, o no usarlo nunca, crear dos partidas y que cada uno fuese servidor de una, aunque esto es imposible, ya que el net.device es de sólo lectura, pero lo intentamos, queriendo hacer una especie de "pásala" con el servidor (en cada frame, el servidor pasaba a ser el otro ordenador, vamos, un caos), pero nada funcionó. Hasta que por un pequeño error de programación, más que un error un tiempo muerto en la ejecución de un programa, hallamos la solución.



Tratamos de crear dos partidas de red, cada usuario servidor de su propia partida, y que se cruzasen los datos. Para ello debían de ser servidor de una y cliente de otra. Para hacerlo, ejecutábamos el programa. Si no había ninguna partida creada, entonces se creaba una, y se esperaban quince segundos para entrar en la otra. Antes de estos quince segundos, debía de entrar el segundo jugador, que entraría como cliente del primero, creando inmediatamente otra partida en la que debía de ser el servidor (por supuesto, todo este galimatías no funcionó). Pero ocurrió algo increíble: antes de que pasasen los quince segundos para que el jugador uno se incorporara al juego ¡¡la red funcionaba!!.

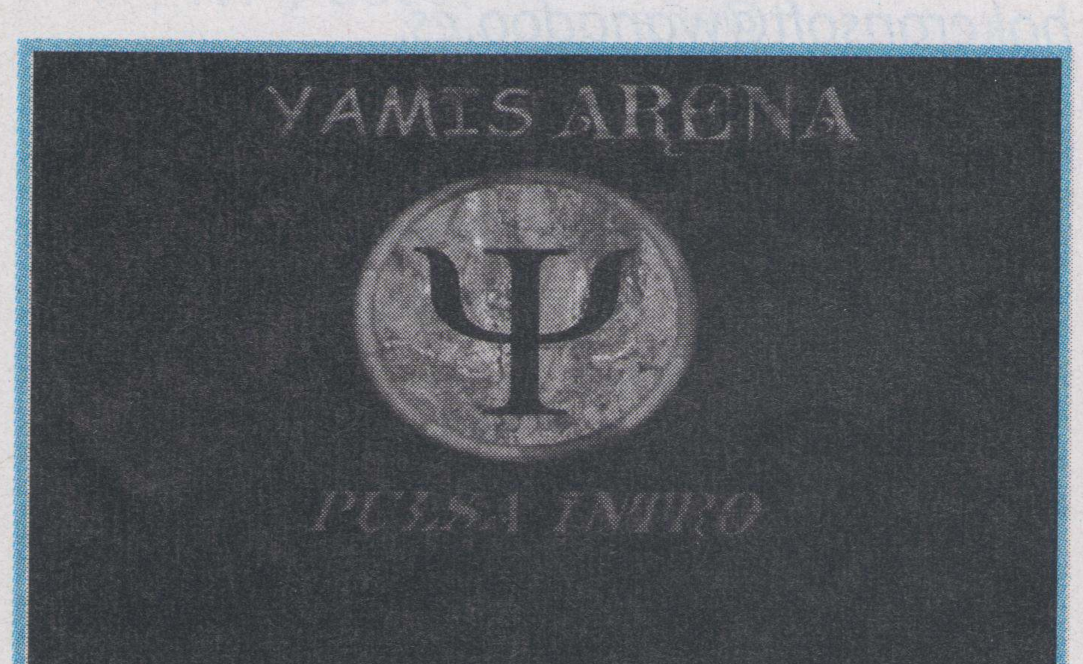
La verdad es que todo esto nos resultó bastante raro, y aún ahora seguimos sin estar muy seguros de su funcionamiento, pero parece ser que hay otro error en las funciones de red, que parece ser el siguiente: cualquier estructura que tenga la misma longitud que la que se crea para jugar en red se pasa automáticamente por la red, siendo legible el campo cero de la misma para todos los jugadores, como si del servidor se tratase. Y aquí radica el secreto: crear una estructura para cada jugador (ej: net1, net2...) y que cada uno de ellos utilice el campo cero para introducir sus datos (parece ser que el primero que introduzca datos, la hace suya, por lo que los demás no pueden modificarlo directamente, aunque no hemos tenido posibilidad de probar con más de dos jugadores, por lo que no podemos afirmar nada al respecto). En fin, esto es todo lo que sabemos hasta ahora. Esperamos que los cientos de Divmaníacos que hay por el mundo amplíen y corrijan nuestras observaciones.

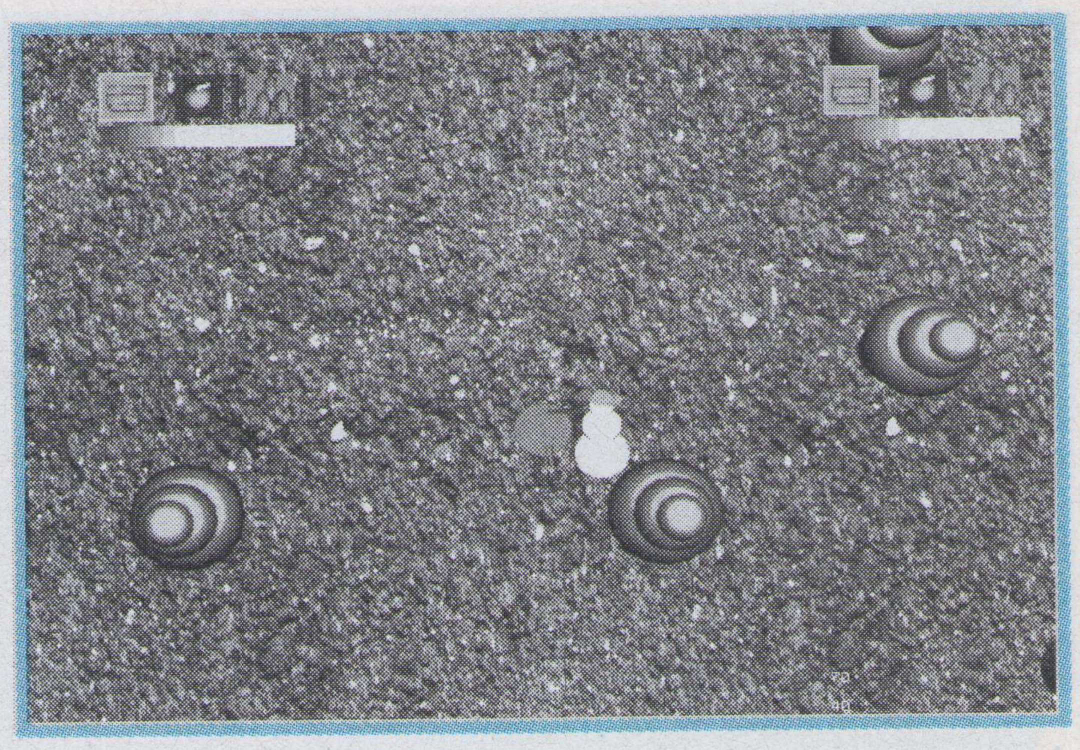


YAMIS ARENA

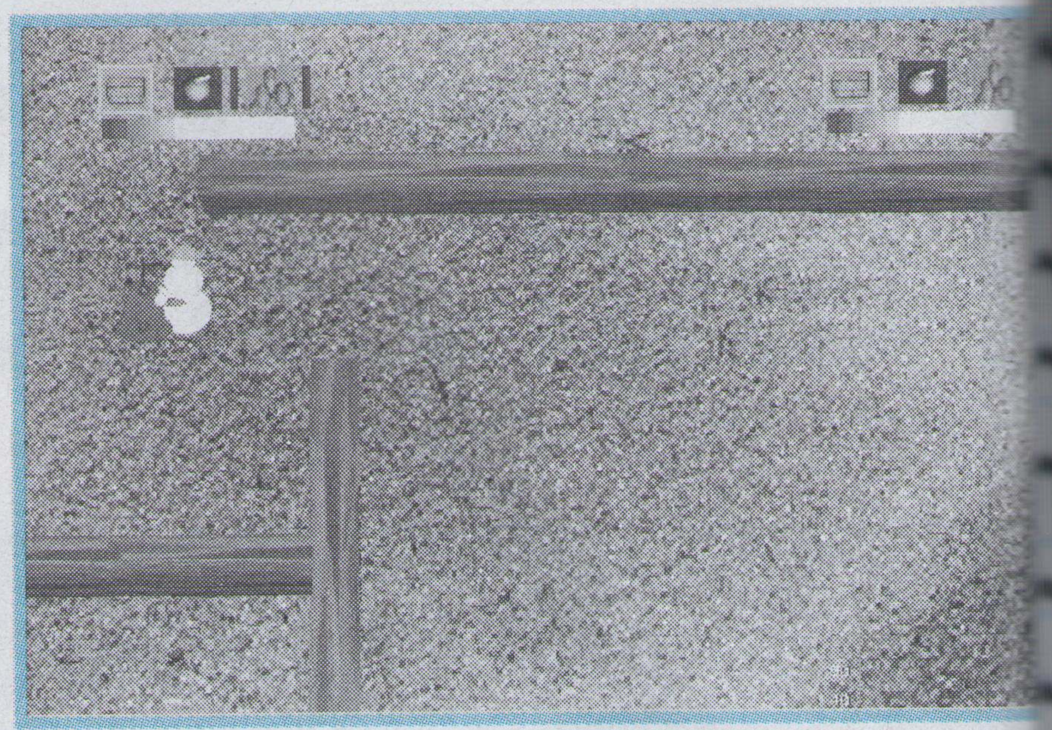
El juego es un arcade de uno contra uno, y está diseñado para dos jugadores, así que no le pidáis mucho al modo un jugador, ya que es bastante simple. La verdad, la idea de programarlo nos surgió desde que vimos el primer Div. Queríamos diseñarlo de nuevo, y ampliar las posibilidades de juego con más armas, scroll y demás. Así que, cuando salió Div 2, con posibilidad de programarlo en red, tras pensarlo detenidamente, y tras programar el *Mr Bones... O vete al infierno*, nos pusimos manos a la obra. La idea es bastante simple, pero el juego engancha bastante, sobre todo cuando dos colegas se "pican" a ver quién gana (aunque la verdad, con tan pocas armas, las estrategias y demás se agotan rápidamente, en la versión final habrá muchas más, podéis comprobar algunas de las que llevará en el *Japi quiler yamis*, la primera parte).

El juego, al ser una beta, tampoco está muy depurado, pero algunas partes, que parecen que podrían estar más depuradas, están así a propósito. Por









ejemplo, un proceso para cada jugador y para la máquina. En la versión final estamos por hacer un proceso por personaje. Sí, habéis leído bien, uno por personaje. ¿La razón?, muy simple: velocidad. Nada más con tres personajes, la cantidad de sí condicionales (IF) que incluye el proceso es enorme, tanto es así que sin ellos la extensión del proceso se- reduciría a más de la mitad. Esto influye también en la velocidad, ya que ha de hacer numerosas comparaciones y verificaciones que ralentizan la acción del juego. Cuando tenga ocho o diez personajes más el rendimiento del juego puede reducirse considerablemente, y todo por hacer un programa más óptimo y legible para el programador. Lo importante, al menos para nosotros es el juego, y si tenemos que hacer un código más grande para que el juego sea rápido, pues lo hacemos (podrían tomar ejemplo esas grandes empresas de por ahí, que sacan algunos juegos que requieren una máquina excesivamente elevada para lo que ofrecen). Ah, esperamos que disfrutéis con la música, cien por cien original, compuesta por Carlos y Miguel. Las incluimos también como pistas de audio.
Bueno, por nuestra parte nada más. Esperamos que el tutorial para programar en red que hemos preparado os aclare todas las dudas sobre la red. Si tenéis alguna pregunta podéis escribirnos a *bokeronsoft@wanadoo.es*.

CÓDIGO FUENTE

(como siempre lo encontraréis completo en el CD-Rom)

```
PROGRAM yamis;

GLOBAL

// variables del juego para la red
STRUCT net1[1];
    vida;
    x_tio;
    y_tio;
    tio_listo;
    tio;
    endisparo;
    velocidad;
    ult_direccion;
    graf;
    selec;
    ind;
    arma;
    bala;
    bomba;
    bazooka;
    ok;
END;
STRUCT net2[1];
    vida;
    x_tio;
    y_tio;
    tio_listo;
    tio;
    endisparo;
    velocidad;
    ult_direccion;
    graf;
    arma;
    bala;
    bomba;
    bazooka;
END;
num_red;
jred;
tio1_cogido;
tio2_cogido;

// variables jugador
anim1[]=2,3,4,5,6,7,8,9,10,11,12;
anim2[]=14,15,16,17,18,19,20,21,22
,23,24;
anim3[]=26,27,28,29,30,31,32,33,34
,35,36;
ani1[]=1,2,3,4,5,6,7;
ani2[]=9,10,11,12,13,14,15;
ani3[]=16,17,18,19,20,21,22;
obstaculo;
zona;
fpg1;
arma1;
arma2;
redi_1;
redi_2;

// variables selecci¢n

op_ppal;
ind;
fpgisla;
x1_nombre;
y1_nombre;
x2_nombre;
y2_nombre;

// variables tio 1
tio1;
vel_x_1;
vel_y_1;
vida_1;
vida_total1;
vida_jug1;
cor_x;
cor_y;
x_tio;
y_tio;

// variables tio 2
tio2;
vel_x_2;
vel_y_2;
vida_2;
vida_total2;
vida_jug2;
x_tio2;
y_tio2;

// Regiones en las que vamos a dividir
la pantalla
region1;
region2;

// Variables de sonido
gong;
boom;
otraarma;
blanco;
dispara;
metralla;
```

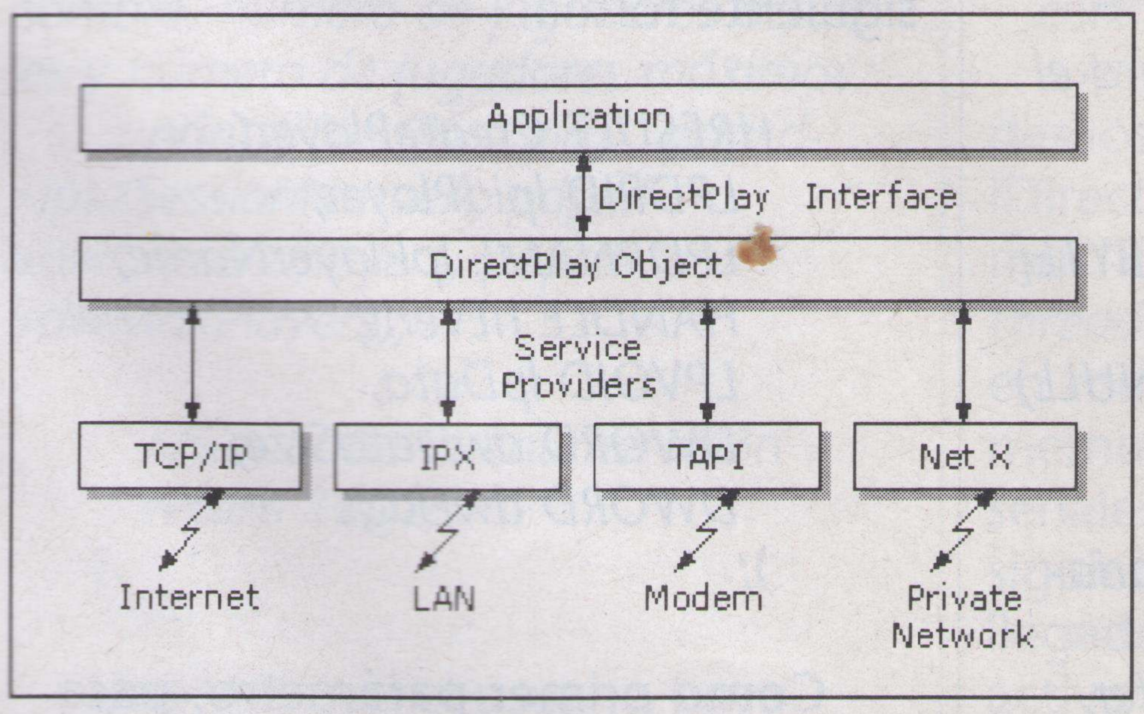
Introducción a Direct Play (1ª parte)

Múltiples posibilidades

En esta entrega analizaremos en funcionamiento general de Direct Play, el módulo de DirectX encargado de simplificar la programación de las comunicaciones y modos de juego en red.

Para aquellas personas que hayan trabajado en un sistema operativo UNIX, el concepto de *socket* seguro no les es desconocido. Se trata de abrir un zócalo en nuestra máquina, asignarle un puerto y "conectarlo" a otro abierto en otra/s máquina/s que también tiene habilitado uno en ese puerto, enviando información por él. Este sistema ha sido durante muchos años el utilizado por miles de programadores, incluso en Windows 9X y NT. Un claro ejemplo son los navegadores Internet Explorer, Netscape Communicator o el mismísimo Quake 3 Arena (en su versión Win32 como Linux).

Este es un mecanismo relativamente sencillo y eficiente para enviar o recibir datos mediante protocolos garantizados (TCP / SPX) o no garantizados (UDP / IPX), estando, incluso, orientado a tareas *multi-thread* (multihilo o multitarea) con esperas pasivas y señales o eventos, a la vez que es posible utilizarlo en varias plataformas salvaguardando la portabilidad y compatibilidad.



Aquí vemos la estructura de los diferentes niveles que utiliza DirectPlay.

Con el nacimiento de DirectX pasó algo parecido que con OpenGL. Se daba soporte a los *sockets* en Windows mediante una API llamada *WinSock*, pero se lanzaba una nueva llamada Direct Play.

Pero esta vez no era un rival en sí, sino que se apoyaba en los llamados *Service Providers*, que no son más que una serie de librerías DLL que utiliza para comunicarse con el dispositivo de red, para enviarle o recibir mensajes de él. Es decir, que esta sub API de DirectX se sitúa en un nivel bastante alto, utilizando otras APIs de nivel medio para comunicarse con la capa baja de hardware.

Su funcionamiento básico se centra en la denominada *sesión*. En alguna parte del menú del juego habrá un "Crear sesión", que indica la creación de un espacio al que se irán uniendo una serie de jugadores. Podríamos definirla como "una colección de jugadores".

Estas *sesiones* pueden ser enumeradas mediante una función de DirectPlay que veremos más tarde. Está permitido crear más de uno por máquina, con lo que es posible habilitar servidores en Internet que alberguen y gestionen más de una.

Esta API es bastante flexible en lo referente a modelos de comunicación: podemos optar entre el clásico y eficiente Cliente / Servidor, modelo distribuido punto a punto o un sistema mixto.

También es posible crear grupos de jugadores, utilizar mecanismos garantizados o no garantizados, utilizar protocolos de encriptación y certificación, soporta conexiones telefónicas, por módem y red local y *direct cable* por conexión serie o null-modem y manejo de grupos de jugadores, muticasting-broadcast, etc...

Y, por si fuera poco, los planes para DirectX 8 contemplan el envío y recepción de mensajes de voz comprimidos por Internet... Todo ello mediante un protocolo propio, el *Direct Play Protocol*.

Además, al ser una API basada en el estándar COM (Component Object Model), mediante el uso de *queries*, mantiene la compatibilidad con todas sus versiones anteriores y permite la fácil actualización a versiones posteriores.

Su única desventaja es que, de momento, a la espera de la prometedora consola de Microsoft, nVidia y otros, la X - Box, API sólo se encuentra disponible para las diferentes versiones de Windows CE / 9X / NT y 2000.

La obtención de una instancia de Directplay

Deberemos poner al principio de nuestro programa C++

```
#include <dplay.h>
#include <dplayx.h>
```

Así como incluir las librerías estáticas DPLAYX.LIB, DXGUID.LIB y OLE32.LIB



Entregas 1 y 2 + 2 CD-Roms
+ 1ª entrega de la guía ColorTone por sólo
995 pts.
5,98 €
Retoque imagen
Convertir las imágenes más simples en las más espectaculares será una tarea sencilla.
Impresión digital
Estudiaremos los métodos más efectivos para que la impresión se ajuste a lo deseado.
Preimpresión
El proceso de impresión es fundamental para trabajar en el mercado editorial.
utoedición
1000 ClipArt Professional
Los mejores ClipArt de informática
1
1000 uentes Profesionales
2
ColorTone
Gratis!
ColorTone™
La guía del Color
Una guía del color que será su referencia básica a la hora de elegir los colores más idóneos para sus trabajos. Una ocasión única para adquirir una publicación independiente de gran valor y de forma completamente gratuita con cada número de la colección.
Prens@ Técnic@
de libros y publicaciones
PRENSA TÉCNICA
C/ Alfonso Gómez nº 42 nave 1-1-2
28037 Madrid
Tfno: 91 304 06 22 • Fax: 91 304 17 97
www.prensatecnica.com
PC CD rom

DIV manía



¿Amas la programación de juegos? Esta es tu revista

Ahora sí que podemos decir que estamos inmersos en un verano abrasador. Pero nosotros seguimos al pie del cañón para traeros vuestra revista preferida que, como suele ser habitual, hemos tratado de hacer con todo el esmero que se merece. Eso a pesar de que sabemos que a muchos no les gustará de cualquier modo y que siempre encontrarán motivo para criticarnos.

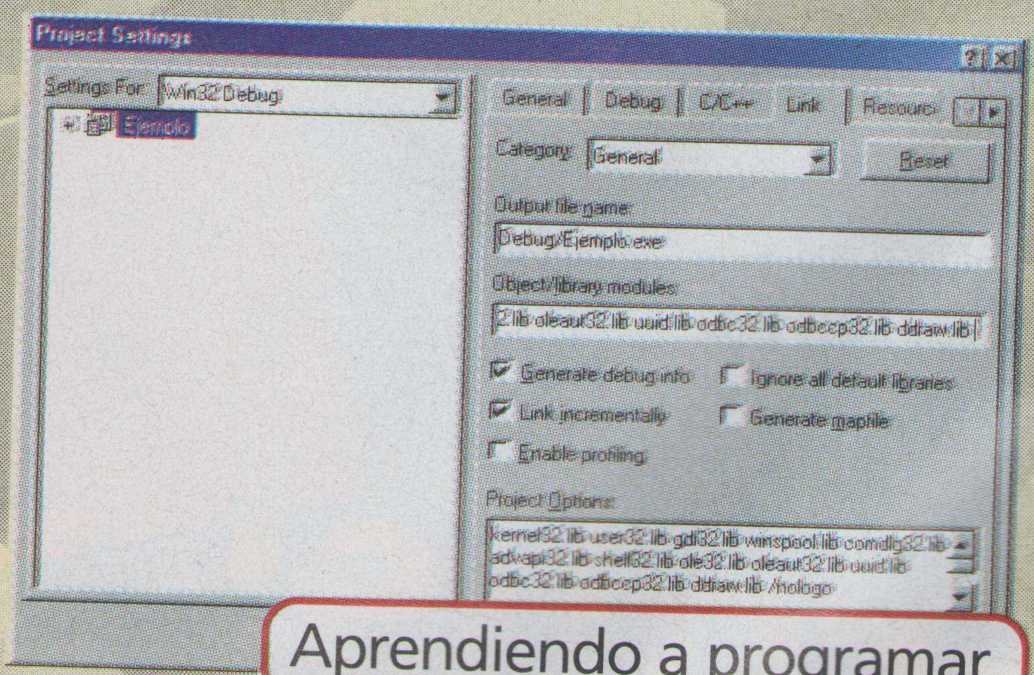
De este mes nos gustaría destacar alguno de los contenidos que hemos recogido. En primer lugar, como no, el extenso reportaje que hemos dedicado a Stratos, una asociación de sobra conocida por todos vosotros que está llevando a cabo una labor encomiable.

Por otro lado, hemos querido volver la mirada hacia Internet y darle la importancia que se merece. Por ello os traemos un reportaje especial enfocado al tema y dos versiones demo de los mejores programas que se pueden encontrar en la red para la creación de páginas web. Esperamos que os gusten y os resulten útiles.

Por otro lado y siguiendo con el contenido del Cd, os traemos dos programas que tiene mucho que ver con Div y que sin duda os van a interesar. Se trata de Dark Basic y MUGEM, un programa del que además tenéis un amplio reportaje dentro de nuestras páginas.

Por lo demás, nuestro contenido es el de siempre y, como siempre (valga la redundancia) esperamos que sea de nuestro agrado. Muchas gracias; nos vemos a la vuelta de vacaciones.

Año 2 · Número 9



Aprendiendo a programar

Director: Mario Luis
mluis@prensatecnica.com

Director Editorial
Eduardo Toribio
etoribio@prensatecnica.com

Director Técnico
Fernando Escudero
fescudero@prensatecnica.com

Coordinador de Redacción
A. Tec. Editorial
Oscar Condés
gover@prensatecnica.com

Departamento de Redacción y colaboradores
Alfredo del Barrio, Antonio Marchal, José A. Migens, Pablo Trinidad, Santiago Orgaz, Fermín Vicente, Emilio Llamas, Ramón de España, Daniel García, Héctor Zapata, Sergio Cánovas y Santiago García.

Departamento de Arte
Francisco Calero

Departamento de Maquetación:
José Antonio Gil, Antonio García Tomé, Jimena Mas, Inmaculada Vaquero, Beatriz Fernández, Lino Herrero y Jesús Aguado

Portada: Francisco Calero

Departamento de Publicidad:
Marisa Fernández, Sonia Glez.-Villamil.

Coordinador de publicidad: Raúl Aguado

Departamento CD-Rom:
Eugenio García y Enrique Roldán

Servicio Técnico CD-Rom:
Auxiliadora Díaz
Horario de atención: tardes 16 - 18 h
E-mail: stecnico@prensatecnica.com

Departamento de Producción:
Ana Domínguez, Gema Alonso, Mª Carmen García.

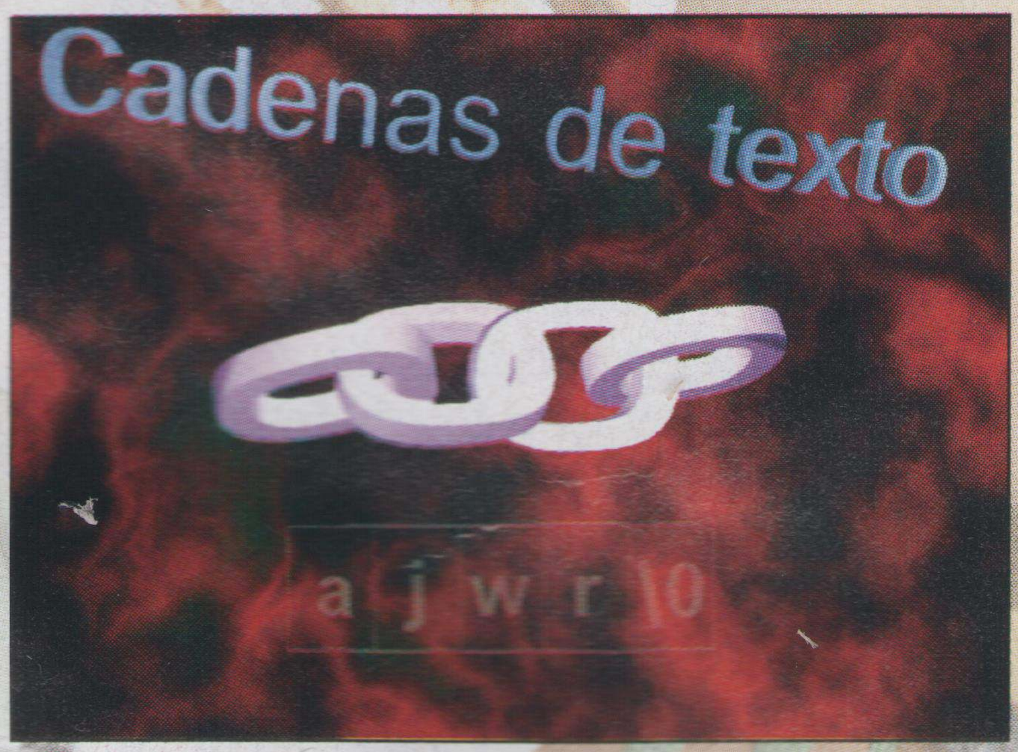
Departamento de Suscripciones:
Sandra Fernández
suscripciones@prensatecnica.com

Secretaría de Redacción:
Cori García Alonso

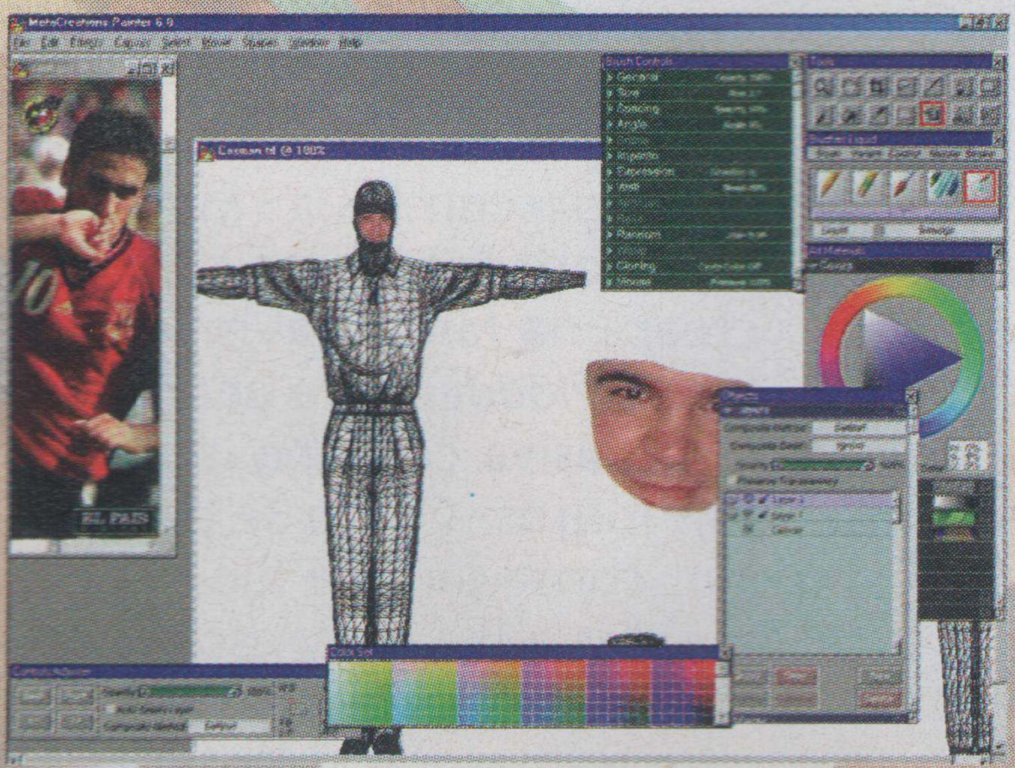
Departamento de Administración:
Juan López, Olga Peña

Redacción, Publicidad y Administración
c/ Alfonso Gómez 42. Nave 1.1.2

Sumario







Programas del lector



PREMIAMOS LOS MEJORES JUEGOS DE LOS LECTORES

DIV ha creado toda una escuela dentro del mundo de la programación. Todos los que se han acercado a la herramienta han sido capaces de programar. Por ello realizamos un concurso entre los lectores que hayan realizado juegos con DIV. Os ofrecemos un primer premio de 25.000 pesetas y dos accésit de 20.000 pesetas.

¿QUÉ ES DIV GAMES STUDIO?

DIV Games Studio es una herramienta de programación que facilita en gran manera nuestra inmersión en el software de entretenimiento. Es el primer entorno profesional que permite realizar videojuegos con fines comerciales sin necesidad de un pago adicional. Con el carné de desarrollador incluido se permite el desarrollo de cualquier tipo de juegos y su libre venta y distribución.



Madrid 28037 España
Tfno: 91 304 06 22
Fax: 91 304 17 97
Si llama desde fuera de España, marcar (+34)
prensatecnica@prensatecnica.com
www.prensatecnica.com
Horario de atención al público: de 9 AM a 7 PM ininterrumpidamente

Edita: Prensa Técnica
Consejero Delegado: Mario Luis
Directores Editoriales: Eduardo Toribio y José Henríquez
Director Financiero: Joaquín González
Directora Publicidad: Marisa Fernández
Fotomecánica: Prensa Técnica
Impresión: Printerman
Duplicación del CD-Rom: M.P.O., Servicios Ibéricos, Grupo Cóndor
Distribución: SGEL. Avda Valdelaparra, 29 Alcobendas. Madrid

Depósito legal: M-11413-1999
AÑO 2 • NÚMERO 9
Copyright 30-11-00
PRINTED IN SPAIN

¿Se puede criticar un juego sin haber jugado con él?

Opinando



Seguimos haciéndonos eco de las cuestiones que nos plantean nuestros lectores. Aunque en principio la carta de este mes se sale del ámbito de la programación, su contenido nos ha parecido más que interesante para todo aquel que esté interesado en este mundillo nuestro, el de los videojuegos.

¿Se puede criticar un juego sin haber jugado con él? Ésta es la pregunta que me hice hace poco cuando vi en una revista la crítica sobre el ya popular juego *Los Sims*. El juego era calificado como muy edificante y moral, lo cual podía incitar a los padres a comprarles el juego a sus hijos. No tengo ningún problema con eso, me parece bien.



Cuando el juego llegó a mis manos, lo cargué en mi ordenador para ver realmente de qué trataba y el susodicho "me enganchó". Al principio todo iba bien; creé una familia, la metí en una casa, les di trabajo y me puse a explorar las opciones que el juego daba. Al cabo de quince minutos comenzó lo mejor: los niños pernoctando mientras el padre, un honrado policía, dormía en la calle cuando tenía que estar trabajando. Eso me motivó para crear una familia antisocial, y lo logré: los bomberos me multaron tres veces por falsa alarma; a uno de los niños me lo llevaron a la academia militar por no acudir a clase; otro se pasaba el día en el mueble bar; conseguí que ardiese un microondas (y eso no lo he conseguido todavía en mi casa); me echaron de tres trabajos y me quedé sin amigos... todo ello amén de tener la casa llena de meadas y basuras.

Tras ésta primera toma de contacto con los atributos no mencionados en las críticas decidí crear una familia al más puro estilo *Los Flodder* para acompañar esta misiva con una imagen de mis logros. Como se puede observar, conseguí que hiciesen cuatro fogatas a la vez, y eso que no era la noche de San Juan, llené la casa de basuras, sólo puse una cama para cinco personas (la que falta era un niño que me lo mandaron a la academia militar), y no sólo no tenía ni un amigo, es que nadie quería venir a mi casa. Yo esto lo conseguí hacer en poco más de media hora. Para que las cosas vayan bien, una de dos: o estas usando los trucos del juego, o sólo has jugado cinco minutos.


Otro caso parecido es el del mundialmente conocido "Carmageddon" y su saga. En las primeras críticas, cuando en España sólo estaba la versión demo, decían que era un juego violento. ¡Vale! Lo entiendo. No es normal que atropelles gente. El problema llegó más adelante cuando tachaban al juego de sanguinario y demás porque decían que el juego se basaba en atropellar gente y que también podías atropellar a embarazadas. El juego no va de eso. El juego es un simulador de coches de carreras y tu misión es acabar la carrera. Lo que hagas aparte ya es cosa tuya. Encima, tras pasarme el juego y extensiones unas nueve veces, todavía no he conseguido atropellar a ninguna embarazada, porque no la encuentro. Eso sí, he atropellado a Papá Noel. También he conseguido pasarme el juego sin atropellar a nadie, cosa que decían que era imposible en las críticas. Así el juego no tiene gracia, pero lo hice.

Tras estos sucesos, y otros más, he llegado a dos conclusiones:

1ª- Si un juego violento lo coge un puritano y juega a su modo, el juego no será violento. Si un juego puritano y moralista lo coge un demente esquizoide, el juego será totalmente violento y antisocial.

2ª- Dejad que escriban las críticas los que saben, no el Opus Dei.

Unai Olabarri.


ENEMY
INFESTATION
Sólo
18,00 €
PC CD rom
2.995
Ptas
Manual en Castellano
PC CD rom
COMPATIBLE
WINDOWS 98
Digital Dreams
MULTIMEDIA
ripcord
GAMES
Distribución Argentina: Take Off Multimedia • Pueyrredon 495
Ramos Mejía CP 1704 • Bs As • Argentina
Digital Dreams Multimedia
C/ Alfonso Gómez, 42, nave 1-1-2
28037 Madrid (España)
Tél: +34 91 304 06 22
Fax: +34 91 304 17 97
www.enemyinfestation.com

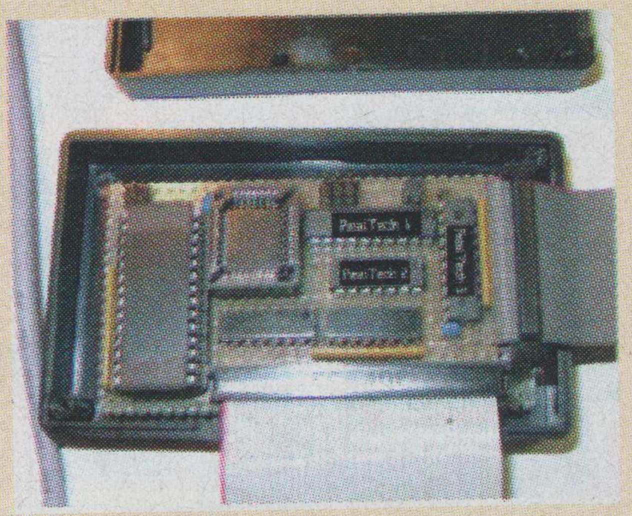
Al asalto de la Dreamcast

Los piratas atacan de nuevo

Nada en este mundo parece estar a salvo de los hackers informáticos y hace tiempo que la nueva consola de Sega estaba en el punto de mira de este colectivo, sólo era cuestión de tiempo que la protección anticopia de la Dreamcast saltase por los aires. Aunque tiempo han tardado, nada menos que un año, en conseguir burlar las barreras de protección de los diseñadores de la consola.

Los GD-Roms de la Dreamcast están escritos en dos sesiones distintas y su capacidad de almacenamiento era superior a los CD-Roms normales, unos 1024 Mb frente a los 650 Mb normales.





A pesar de estas medidas, un grupo de piratas ha realizado un complejo aparato bautizado como Dreamcast Debug Handler (DDH) que conectado a un ordenador y a la Dreamcast permite copiar los juegos en el PC y desde allí piratearlos como si de un CD-Rom corriente y moliente se tratase. Lo único que hará falta para que estas copias ilegales funcionen es meter antes un disco de arranque que engañe a la consola.

El Gran Hermano salta al PC

Un juego que aprovecha el tirón del concurso televisivo



Infogrames ha publicado un título basado en el concurso de Tele 5 Gran Hermano. El juego te llevará a intentar superar la numerosas pruebas a las que son sometidos los concursantes y te obsequiará con las imágenes más impactantes del programa real. Los personajes están caricaturizados y son los mismos que los del programa.

Parece que este juego ha sido editado en otros países donde se ha puesto en antena el concurso y ha tenido gran éxito.

Viendo las imágenes de este juego llegamos a la conclusión que a nosotros nos han llegado mejores títulos para participar en el concurso de DIVmanía, lo que nos ha llevado a pensar lo injusta que es la vida. Seguro que este juego del Gran Hermano vende miles de copias, pero en fin siempre nos queda la dignidad.



PS One

La nueva consola de Sony



Está claro que Sony no se duerme en los laureles. Por fin se ha desvelado la sorpresa que se andaba esperando de manos de la multinacional nipona. Sony ha confirmado que va a entrar en el mercado de las consolas portátiles con una versión reducida de su clásica PlayStation. Se llamará *PS One*, ocupará tan sólo un tercio del espacio de la original y estará a la venta en Japón, al precio de 15.000 yenes (unos 142 dólares o 24.000 pesetas). A Norteamérica y Europa llegará este otoño, posiblemente a partir de septiembre por lo que adelantará a la PlayStation 2.

Se esperan producir y vender alrededor de 6 millones de unidades de aquí al 31 de marzo del 2001, y además de las funciones básicas de la PlayStation de siempre también tendrá capacidades de conexión a través de teléfonos móviles desde el próximo invierno, algo que también llegará a PSX y PS2, permitiéndoles descargar todo tipo de información y software así como intercambiar partidas y datos con los amigos. Pero lo mejor de todo es que será totalmente compatible con los más de 800 títulos existentes en la actualidad para PlayStation.

Actualizando a los clásicos

Doom conocerá una tercera parte



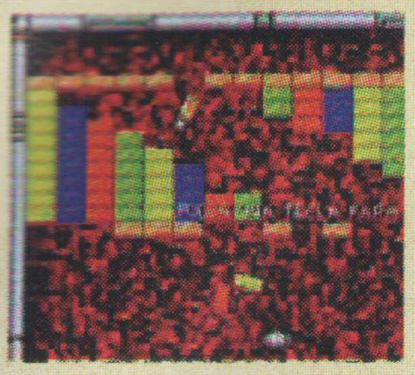
Si hace poco se anunció que *Wolfstein 3D* conocerá una nueva puesta de largo, parece que tampoco ha querido desaprovecharse la archiconocida licencia *Doom* para actualizar un juego que hizo las delicias de millones de jugadores allá por el principio de los años 90. Si el primero lo desarrollará una empresa externa, de la tercera parte de *Doom* se va a encargar la mismísima Id con John Carmack a la cabeza, uno de los padres de la criatura junto con John Romero.

No sabemos nada del nuevo proyecto, excepto que será un juego eminentemente para un solo jugador y van a ponerse manos a la obra enseguida. Aunque de los primeros creadores sólo queda Carmack en Id Software actualmente, este personaje es suficiente como para tener la vista puesta desde ahora en todo lo relacionado con *Doom III*. Y, por cierto, hay que decir que a Carmack le ha costado bastante convencer a sus socios de la viabilidad de un nuevo *Doom*, pero al final el gurú se ha llevado el gato al agua y se ha salido con la suya.

Fénix se actualiza

Se incorporan nuevas funciones

El lenguaje pseudo interpretado diseñado para realizar cómodamente juegos en dos dimensiones a base de sprites, Fénix, ha incorporado al código de CVS un juego de funciones que permitirá realizar sin ningún problema las operaciones llamadas "blendops" con los gráficos de 16 bits. Estas operaciones no modificarán el gráfico en memoria, sino que afectarán únicamente a la rutina de dibujo. Parece que poco a poco Fénix va tomando forma, esperemos que dentro de poco sea una herramienta completamente operativa.



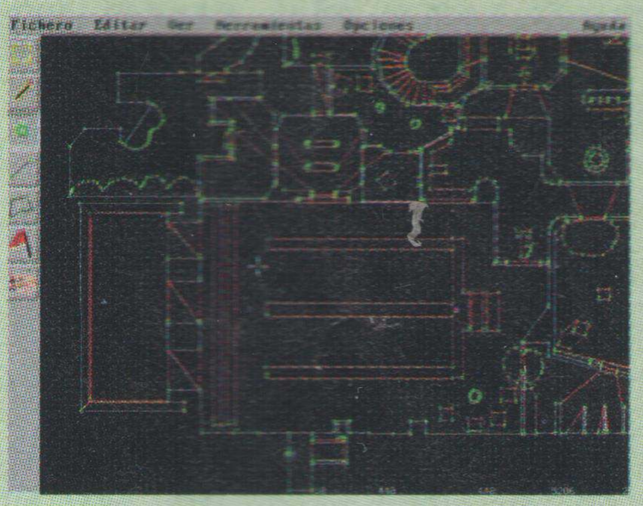


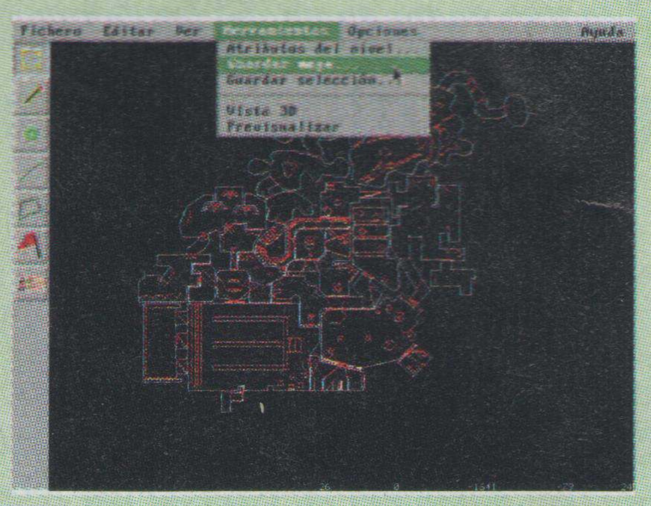
De todas formas, es conveniente que te pases de vez en cuando por la página web de Fénix para estar al día (http://www.arrakis.es/~jlceb/fenix/index.html).

Mapas en modo 8

Pronto estará disponible un creador de mapas

En la siguiente dirección web: http://www.sferasoft.org/ermakina/m8dcap.html, podréis encontrar noticias sobre lo que parece ser un futuro creador de mapas para el modo 8 de Div2. El autor del proyecto es T_Machine y ha puesto unas cuantas imágenes realizadas con la nueva herramienta, que llevará el nombre de Mode8Designer. Ánimo y pasaros si podéis por su página.





Los mejores del E3

¿Quién no quiere uno de estos premios?

Como cada año, no ha habido sorpresas. Los premiados han recibido el reconocimiento por su gran éxito comercial. Los juegos más recompensados, han sido: *Los Sims*, *Final Fantasy VIII* y *Age Of Empires 2: Age of Kings*, con tres premios para cada uno. Sorprende un poco que Quake III Arena no se haya llevado nada, pero que le vamos a hacer. Vamos a ver los galardones con más profundidad:

Premios Artísticos:

Premio a la mejor Dirección Artística: *Final Fantasy VIII* (Square Soft/Playstation).
Premio a la mejor Animación: *Final Fantasy VIII* (Square Soft/Playstation).
Premio al mejor Sonido: *Medal of Honor* (Dreamworks/Playstation).
Premio a la mejor Composición Musical: *Um Jammer Lammy* (Sony/Playstation).
Premio al mejor Diseño: *Los Sims* (Maxis/PC).
Premio al mejor Grafismo: *Unreal Tournament* (Epic Megagames/PC).
Premio al mejor Gameplay: *Los Sims* (Maxis/PC).
Premio al mejor Guión: *Age of Empires 2: Age of Kings* (Ensemble/PC) y *Thief: The Dark Project* (Looking Glass/PC).

Premios Consoleros:

Mejor juego de acción del año: *Crazy Taxi* (Sega/Dreamcast).
Mejor juego de rol/aventuras del año: *Final Fantasy VIII* (Square Soft/Playstation).
Mejor juego de lucha del año: *Soul Calibur* (Namco/Dreamcast).
Mejor juego de carreras del año: *Star Wars: Episode One Racer* (Lucas Arts/Nintendo 64).
Mejor juego familiar del año: *Pokémon Snap* (Hal Labo/Nintendo 64).
Mejor juego de deporte del año: *Knockout Kings 2000* (EA Sports/Nintendo 64).
Mejor juego de consola del año: *Soul Calibur* (Namco/Dreamcast).

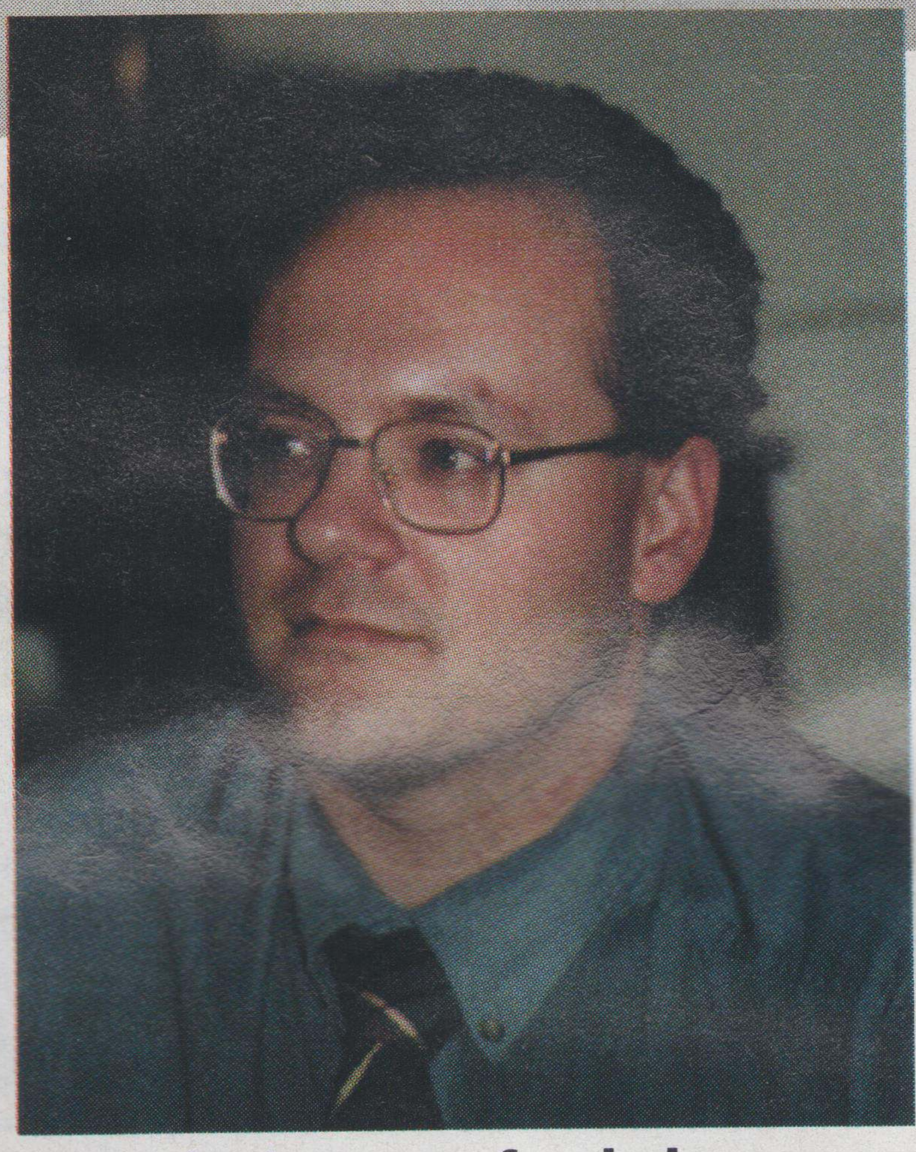
Premios para PC:

Mejor juego Online del año: *Everquest* (Verant Interactive).
Mejor juego de acción del año: *Half-Life: Opposing Force* (Valve Software).
Mejor juego de rol / aventuras del año: *Asheron's Call* (Turbine).
Mejor simulador del año: *Microsoft Flight Simulator 2000 Professional* (Microsoft).
Mejor juego de deporte del año: *FIFA 2000* (EA Sports).
Mejor juego de estrategia del año: *Age of Empires 2 : Ages of Kings* (Ensemble).
Mejor juego para PC del año: *Age of Empires 2: Ages of Kings* (Ensemble).
Mejor juego del año: *The Sims* (Maxis).

Stratos

Un proyecto con futuro

Este mes haremos un reportaje a la asociación de desarrolladores de Stratos. Ya tuvimos un reportaje sobre ellos en otro numero de esta revista; pero, cuando sucedió esto, DIV era una herramienta que acababa de nacer, como quien dice. Ahora, después de haber pasado un tiempo prudencial, volvemos a fijarnos en Stratos para ver de qué manera ha repercutido DIV en la asociación.



Antonio Arteaga, fundador y coordinador de Stratos.

Antes de entrar en materia, haremos un pequeño resumen de lo que es la asociación, su origen y evolución para aclarar conceptos, recordar o mostrársela a todos aquéllos que la desconozcan. Así que, para empezar, haremos un poco de historia.

Orígenes y evolución

Stratos es la mayor asociación hispana de desarrolladores. Engloba a más de 400 grupos de desarrollo y está formada por miles de personas. Los fundadores de Stratos son Antonio Arteaga Pérez y José Carlos García Martín que, tras darle muchas vueltas a la idea, decidieron crear una entidad común que sirviese de punto de encuentro de todos los desarrolladores y que sirviese para ayudarlos y potenciar su labor.

Stratos surge en 1997 con la idea de ser un punto de encuentro para desarrolladores

Así, a principios de 1997 surge Stratos, en principio con sólo siete grupos y abriendo su actividad en Internet con una pequeña página web. Los comienzos fueron difíciles, sin embargo, la publicación de los primeros juegos (*La Luz del Druida* y *Arakhas el Oscuro*), seguida de títulos posteriores como *Twinfire, Ping Pong* y *Bricks de Luxe* (todos en 1997), comenzó a dejar huella en la comunidad de desarrolladores españoles, que ya empezaron a tomar en cuenta a Stratos, como una puerta abierta a sus proyectos y como el lugar donde encontrar el apoyo que les faltaba.

En 1998 se publicaron *Osarium, Space Plumber, Oxide, La Fuga, Twin Lazers, Witch Frog, Fatal Jack* y *Down to Hell*, lo que sirvió para dar más nombre a los grupos desarrolladores (Island Dream, iLogic, Nerlaska, Exelweiss...). En esta época el lema de la asociación era "Juntos podemos conseguirlo todo. Por separado es prácticamente imposible".



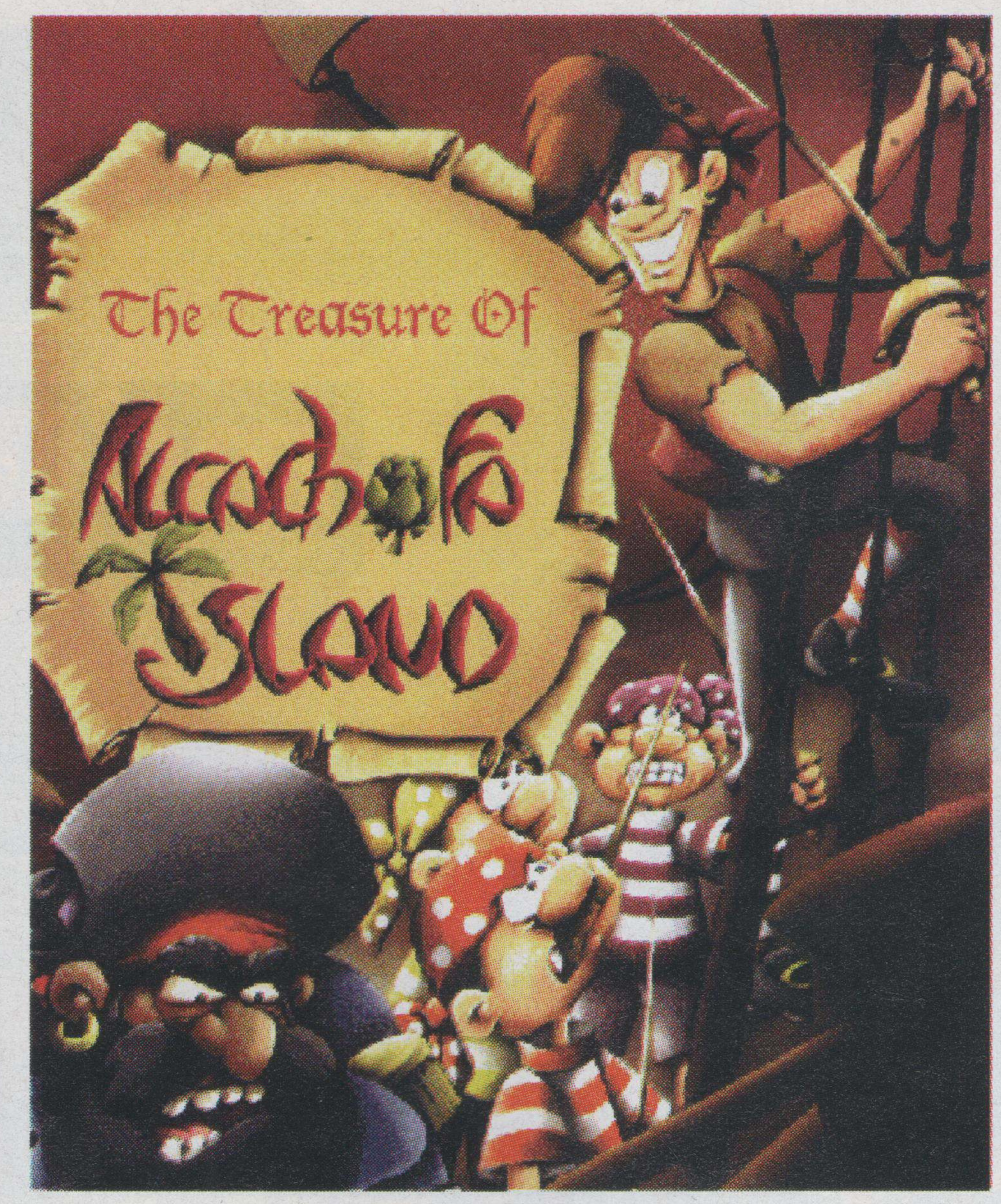
Ancient Stories: Black Lands **(Exelweiss), novedoso juego de rol y estrategia con personajes en 3D.**



Downstone **(Brainstorm Interactive), un muy adictivo arcade.**



***Super Tour 2000*, creado por Miguel Angel Haza, es un excelente simulador del deporte de la bicicleta en DIV.**



***The Treasure of Alcachofa Island* (Alcachofa Soft), una increíble aventura de piratas.**

Y así fue. En 1999 se produce el mayor crecimiento de Stratos, que pasa a superar los 300 grupos miembros y rebasa las fronteras españolas para dar cabida a los desarrolladores latinoamericanos. La práctica totalidad de países de Hispanoamérica cuenta con algún miembro de Stratos y, de hecho, se está ultimando la creación de Stratos Latina, una división de la asociación que, con su propia página web, estará centrada en los miembros y actividades de los países americanos.

Paralelamente, la página web de Stratos (*http://www.stratos-ad.com*) ha venido sufriendo una evolución considerable, pasando de tener unas cuantas secciones a convertirse en prácticamente un portal de desarrollo donde se han tenido en cuenta todos y cada uno de los sectores y áreas de interés de la comunidad de desarrolladores hispanos. Además, la existencia del canal *#programacion_stratos* en el IRC Hispano también sirve como punto de encuentro para conversar, pedir ayuda o intercambiar impresiones.

En este año salieron a la luz títulos como *Mr. Tiny, Luridland, Sleepwalker, Astro Assembler, Bricks Gold, Highflyer, Malvinas 2032* o *Karting Racer*. Prácticamente todas las empresas del sector comienzan a acudir a la asociación para conseguir cubrir sus puestos de trabajo y para realizar encargos específicos.

El año 2000 va a suponer un importante punto de inflexión para Stratos. Las previsiones hacen suponer que el número de títulos publicados será muy superior al de años anteriores. *Lookz, Visual Gen, Madd Bomber Xplozion, Theme Puzzle* o *Downstone* son solamente el inicio. Se han firmado acuerdos con multinacionales extranjeras para la comercialización de gran cantidad de productos realizados por miembros de la asociación, y el mercado americano contará con varios títulos en miles de tiendas a principios de este verano.

Los objetivos primarios que se marcaron los fundadores ya se han logrado: poner en contacto a los desarrolladores para afrontar proyectos comunes, tener vías de comercialización para los productos, abrir las puertas de un mercado laboral tan específico como es el de la industria del desarrollo de videojuegos y multimedia.

Ahora surgen nuevos retos que en un principio ni siquiera podían plantearse, como el de convertirse en un ente autosuficiente y con capacidad para poder comercializar por cuenta propia sus propias creaciones, pero que apoyándose en el trabajo diario y, sobre todo, sin perder de vista el objetivo inicial, algún día también se conseguirán.

El año 2000 va a ser un punto de inflexión en la historia de Stratos

DIV en Stratos

En los tiempos que corren, la mayoría de las empresas están optando por aceptar únicamente programas y juegos realizados bajo Windows, con especial predilección por las DirectX y aceleración por



***Cosmos* (Skynarf Software), estrategia por correo electrónico.**



La epopeya de don Primerizo Lata (MDQ Inc.), una divertida aventura del otro lado del charco, Argentina.



Divine & Demonic (Emerge Designs), un tenebroso RPG.

hardware. Visto esto, parecería que DIV no tiene mucho sentido dentro de una asociación que, precisamente, debe estar al día de la demanda del mercado y apostar por las últimas tecnologías.

Pues bien, el coordinador general de Stratos, Antonio Arteaga, piensa que DIV es una herramienta indispensable y, sin lugar a dudas, una de las bases fundamentales de la asociación. ¿Por qué?

DIV es fundamental, ya que sirve para iniciarse a miles de desarrolladores

No todos los grupos de la asociación son profesionales, al contrario. La mayoría de ellos comienzan realizando sus primero pinitos con la creación de pequeños juegos. Y aquí es donde DIV juega un importante papel. Al utilizar este entorno de desarrollo como primera herramienta, los grupos y personas van tomando conciencia de los puntos esenciales de todo desarrollo: diseño integrado, unión de las distintas partes (programación, gráficos, sonido...) para formar un producto final, utilización de rutinas y procedimientos ajenos que más tarde podrán extrapolarse a otros entornos de forma económica y sencilla.

Stratos ha publicado algunos juegos desarrollados con DIV, como es el caso de *Isometric Bomber*. Está claro que no es fácil conseguir un acabado profesional pero realmente la utilización de herramientas más profesionales como Visual C++, DirectX, Glide u OpenGL no aseguran tampoco este acabado.

Se trata más de coger unas ideas y unas costumbres para obtener un producto final. Son muchos los proyectos que se están desarrollando en Stratos utilizando DIV como herramienta. Cuántos de ellos tendrán buena salida es una incógnita, pero sí podemos estar seguros de que la experiencia acumulada durante su desarrollo será importante de cara al futuro.

Proyectos actuales

Pueden definirse tres grupos claramente diferenciados entre los proyectos que constantemente se inician en el seno de la asociación:

- Por un lado, aquellos desarrollos que suponen la primera incursión de los grupos o personas dentro del ámbito de la creación.
- Por otro, las aplicaciones que, contando ya con una mayor solidez y acabado final, pueden tener salida en el mercado pero a través de pequeñas distribuidoras, editoriales, promociones, etc.
- Y, finalmente, los proyectos que pueden calificarse de comerciales, y que están en condiciones de rivalizar con otros productos existentes.

A este respecto, siempre tenemos que tener en cuenta que los grupos están por lo general formados por personas con más ilusiones que respaldo económico. Por tanto, en la inmensa mayoría de los casos es impensable creer que se pueden obtener resultados similares a los de una empresa con un fuerte presupuesto y con decenas de personas asignadas al proyecto.



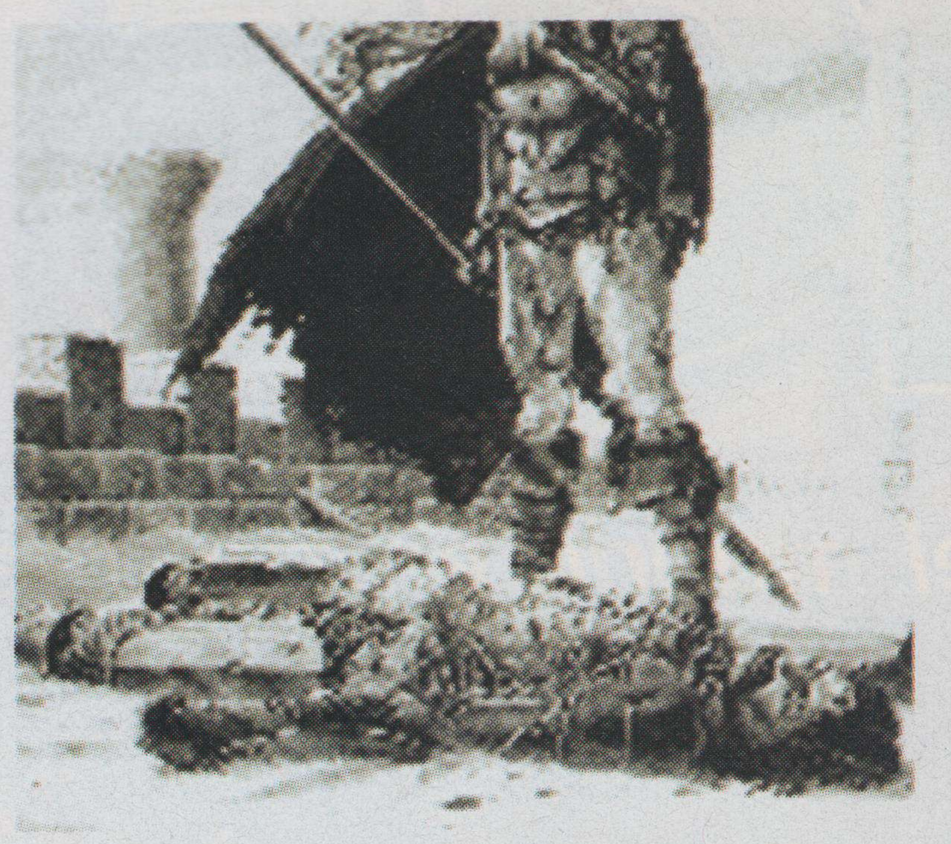
Typhoon (Tao Spain), una recreativa que revolucionará las salas de videojuegos.



Fiebre Pirata (Diego-Films Entertainment), un juego al más puro estilo de las aventuras gráficas.



Mega Games
La revista de videojuegos que pondrá el mundo en tus manos
sólo 595 Pts.
CD ROM Incluido
Mega Games
PC • Playstation • Dreamcast • Game Boy • GBC
1 Juego Completo
Especial juegos
Quake III Arena
Unreal Tournament
Driver
Darkstone
Discworld Noir
Fifa 2000
Trickstyle
Descent 3
Age of Empires 2
Homeworld
Freespace 2
Faraón
GP 500
C&C Tiberian Sun
The Nomad Soul
Reportajes especiales:
Bienvenidos al Siglo XXI
Historia de los videojuegos
• Tiempo de Ocio
• El Manga nos invade
• Consejos para ligar
Especial Verano
Los mejores consejos para pasar un verano a tope



***Thunder Lands* (Cuchipandi Technologies), estrategia por turnos para Game Boy Color.**



***K.O.R.* (CyberGames), un juego 3D realizado en DIV.**

Vamos a citar algunos de los desarrollos que se están gestando en Stratos, pero aclarando que son simplemente eso, una muestra, puesto que el número de proyectos es tan elevado que ocuparía bastantes páginas enumerarlos.

Actualmente, Stratos desarrolla una cantidad muy grande de proyectos

- *The treasure of Alcachofa Island* (Alcachofa Soft). Espléndida aventura realizada por los creadores de *Dráscula* y el *Sulfato Atómico. http://www.alcachofa-soft.com.*
- *Divine & Demonic* (Emerge Design). Tenebroso RPG de ambientación igualmente siniestra, mezcla de *Quake, Resident Evil* y *Diablo. http://come.to/emerge.*
- *The Graveyard* (Emerge Design). Experimento de RPG unipersonal por Internet, en vista isométrica. El ambiente y diseño se basa en el RPG *Diablo. http://come.to/emerge.*
- *Ancient Stories: Blacklands* (Exelweiss). Juego de rol con dosís de estrategia, aventura y misterio, ambientado en la magia, la espada y el honor. *http://www.exelweiss.ivinf.com/*
- *Downstones: The revenge of the Stoonies* (BrainStorm Interactive). Arcade que reversiona los clásicos como *Puyo Puyo 2, Kirby's Avalanche.*
- *Black Sands* (Paranoic Software). Una cárcel abandonada, un grupo de jóvenes estudiantes, sucesos paranormales, una secta satánica... Una mezcla de *Resident Evil* y *Riven. http://www.cyber-jaen.es/web/genomax/index.htm.*
- *Typhoon* (Tao Spain). *Typhoon* es un simulador de vuelo, con el jugador en el interior de una cabina (Recreativa 6DOFXtended). El jugador posee un Acelerador de Gas Progresivo, dos pedales y un joystick que controla el simulador. *http://www.taospain.com.*
- *La epopeya de don Primerizo Lata* (MDQ Inc.). Aventura por un grupo argentino y protagonizada por un gaucho. *http://donprimerizo.mdq.com/*
- *Air Forces* (Cuchipandi Technologies). Juego para Game Boy Color. Metido en la piel de un joven que sueña con ser piloto tendrás que ponerte al control de hasta 16 aparatos voladores.
- *Thunder Lands* (Cuchipandi Technologies). Juego para Game Boy Color. Estrategia por turnos en la que podrás elegir entre Humanos, Orcos, Elfos o Enanos para completar campañas militares.
- *Time Riders* (Morphing Studios). Ambientado en el 2073, un juego en el que siete corredores, con coches ultramodernos, reali-



***Black Sands* (Paranoic Software), un terrorífico juego que mezcla dos títulos bien conocidos: *Resident Evil* y *Riven*.**



***Cyrok* (SilverSky Games), un título repleto de acción.**

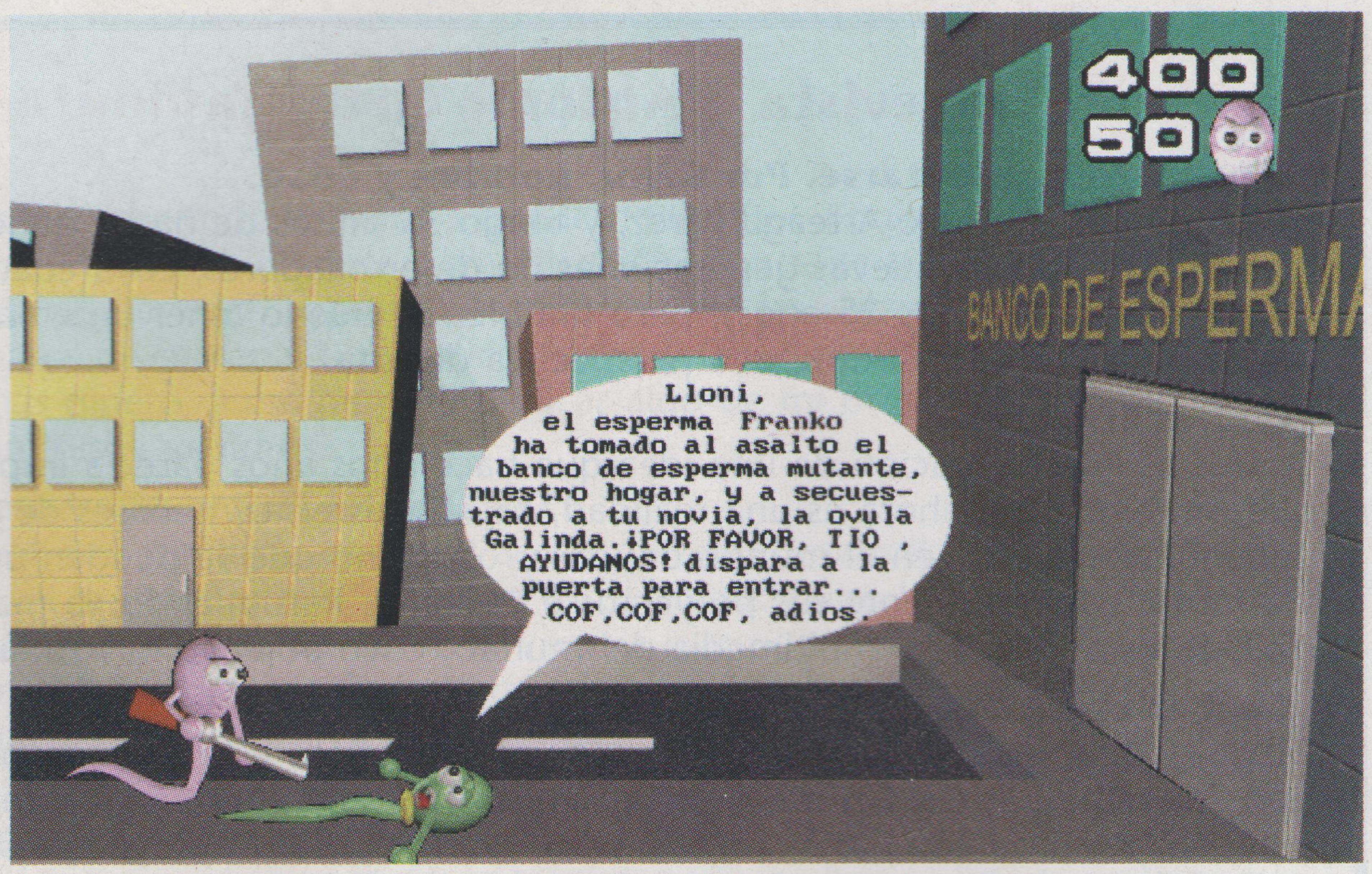
zan competiciones en diferentes épocas de la historia.

- *Fiebre Pirata* (Diego-Films Entertainment). Aventura gráfica con un estilo muy similar al *Monkey Island 3. http://www.fortunecity.com/virtual/video/88/fiebmenu.htm.*

Proyectos en DIV

Como indicamos anteriormente, DIV está muy asentado en Stratos, y gran cantidad de desarrollos utilizan esta herramienta obteniendo en muchos casos resultados verdaderamente vistosos.

- *Super Tour 2000* (Miguel Angel Haza). Simulador de ciclismo en carretera. *http://www.arrakis.es/~mangelh/st2.htm.*
- *K.O.R.* (CyberGames). Se trata de un shooter 3D en primera persona. *http://pagina.de/coolio.*
- *Cyrok* (SilverSky Games). Una civilización extraterrestre se está preparando para invadir la Tierra. Nuestro objetivo es dejarnos capturar para destruir su planeta. *http://club.telepolis.com/jpbenp*



Lloni Be Good **(Devilish Games), divertido juego de aventura/plataforma.**

- *Cosmos* (Syknarf Software). Juego de estrategia ambientado en un universo donde podrán jugar hasta diez jugadores a través del correo electrónico (intercambiando un fichero de datos). Tiene opciones como construcción de naves espaciales, colonizar planetas, gestionar los recursos, luchar, aliarte con ellos... *http://pagina.de/knarf.*
- *Lloni Be Good* (Devilish Games). Una mezcla entre aventura y plataformas, cuyo personaje principal es un espermatozoide que debe rescatar a su novia (la óvula Galinda). *http://members.es.tripod.de/DEVILISH/*

El elevado ritmo de crecimiento de Stratos la consolida como la más grande del mundo en número de miembros

El futuro

El elevado ritmo de crecimiento de Stratos consolida a esta asociación como, posiblemente, la más grande (en cuanto a número de miembros) existente en el mundo. Tras superar con notable éxito sus etapas iniciales y sentar las bases para su futuro, Stratos sigue ampliando sus canales de distribución en todo el mundo, con el objetivo primordial es ayudar y orientar a todos los que desean estar algún día entre los grandes de esta industria.

Se sabe que el trabajo es, sin duda, enorme, pero también es cierto que con la aportación y la experiencia conjuntas se pueden seguir rompiendo barreras y, poco a poco, llevar a la comunidad de desarrolladores hispanos al lugar que se merece.

Antonio Marchal

Cuadro 1. Stratos en Internet

La página de Stratos es: *http://www.stratos-ad.com*

Stratos, como toda entidad que se precie, posee una página en Internet con interesantes contenidos. Nada más entrar nos encontraremos con las noticias del día, que se actualizan constantemente, además de otras muchas secciones.

La página se divide en distintos apartados, según su cometido. En la parte de *Asociación* podremos encontrar información sobre los miembros de la misma, que son un gran número. También en esta sección se podrán ver los títulos que Stratos ha sacado a la venta o ha distribuido. También podremos ver los desarrollos actuales para ver los nuevos juegos que prepara la asociación para sacar en breve. También, cómo no, podremos contactar con la asociación, y podremos encontrar un enlace con Stratos en latinoamérica, ya que la asociación también tiene miembros al otro lado del charco, y la coordinación, debido a la diferencia tanto de lugares como de mercado, es distinta.

Luego pasamos a la sección de *Recursos*, donde podemos encontrar cursos, libros, motores, fuentes, enlaces o descarga. Desde esta sección también entraremos en la parte dedicada a foros o listas. Reseñar en esta parte, la sección de trabajo, ya que la página dispone de numerosas ofertas de trabajo relacionadas con el mundo informático, y más concretamente con el desarrollo de programas, así como de una bolsa, permanente, que se actualiza diariamente. Tanto si buscas trabajo, como si quieres incluir tu curriculum, éste es un buen sitio para hacerlo.

Y llegamos a la sección *Miscelánea*. En ella se agrupan distintas partes, por un lado la revista electrónica *Macedonia*. Así como las distintas entrevistas hechas en distintos medios a la asociación. Por último, también incluye una sección de curiosidades y una galería de fotos, de quedadas, etc.

Finalmente entramos en la sección de *Otros*. Aquí los apartados que nos encontramos son: diseño, colaboraciones y línkanos. Casi todos ellos relacionados con la relación de usuarios con la página.

Además de la página en Internet, semanalmente, si uno lo desea y es miembro de la asociación, recibirá las noticias semanales más importantes mediante un boletín. Éste también se recibirá en la lista de correo de Stratos. Esto se hace así porque no todo el mundo puede visitar su página a diario, donde se encuentran las mismas noticias. Además también disponen de un canal de IRC, es decir, tienen un salón registrado, llamado *#programacion_stratos*, donde podréis entrar desde cualquier servidor hispano, como *irc.jet.es.*

Cuadro2. Entrevista a Antonio Arteaga

Lo primero, presentarse. Por favor, nombre y edad.
Mi nombre es Antonio Arteaga Pérez, y tengo 34 añitos de nada. Aunque debes sentirte joven y poder identificarte con las nuevas generaciones de desarrolladores, cuya edad media es bastante inferior a la mía (entre los 17 y los 25 años), pero también es bueno tener una madurez y experiencia que te permitan observar las cosas desde una perspectiva distinta.

¿Qué es Stratos?
Stratos es la consecución de una idea que hace unos años parecía impensable en el mundo hispano y que, sin embargo, ahora es una realidad. Ya no existen centenares de grupos diseminados que hacen pequeños escarceos en el mundo del desarrollo, sino que se pueden unir bajo un denominador común y afrontar conjuntamente sus proyectos e ilusiones. Resumiendo: la mayor asociación hispana de desarrolladores de juegos y multimedia del mundo, cuyo objetivo es ayudar en todos los aspectos posibles.

¿Cuál es el papel que desempeñas en la asociación?
Dicho de una forma simple, el de coordinador. Prefiero este término porque no me gustan otros del tipo "director", "presidente", "jefe" o similares. Como coordinador, manejo el fichero de miembros (altas, bajas, modificaciones, contactos...), reviso todos los proyectos de la gente, busco vías de comercialización, entablo relaciones con empresas, procuro estar al día de todos los aspectos, etc. Y es una tarea bastante dura, porque va desde aconsejar a un principiante hasta hacer el seguimiento completo de un producto.

¿Cómo te iniciaste en el mundo de la informática?
Los primeros pasos fueron con un Spectrum. Comencé jugando, luego pasé a teclear algunos programas de revistas y, finalmente, acabé haciendo mis propias "cosillas", tanto para Spectrum como para MSX. Y de ahí directamente al PC, ya en GW-Basic, Pascal, Cobol, Clipper, C...
Mi primer trabajo oficial fue de profesor de programación para el INEM, después de programador de gestión, entré como operador de consola en una entidad bancaria, pasé a ser jefe de sala del centro informático de la misma entidad... Prácticamente he tocado todos los aspectos.

Y más específicamente, ¿cómo comenzaste con los videojuegos?
Pues como ya he dicho, tecleando los listados que venían en las revistas de Spectrum. Después empecé a programar mis propios juegos para Spectrum y MSX, luego (ya en PC) realicé dos o tres juegos en Basic, después en Pascal, y mis primeras creaciones ya más aparentes fueron con Turbo C y Watcom (los dos primeros fueron *Snakers* y *Crash*, el último fue finalista de un concurso). Con Watcom hice *Twinfire*, un arcade publicado en Computer Gaming World, y Smasher, vencedor del concurso de PC World. Últimamente, con DirectX, he terminado *Lookz* (licenciado a Crystal Interactive) y estoy ultimando *Theme Puzzle*.

¿Cómo empezaste en la asociación?
Bueno, sería más exacto preguntar "cómo empezó la asociación conmigo". Aunque le llevaba dando vueltas a la idea de crear una asociación de desarrolladores desde hacía tiempo, fue en la entrega de premios de un concurso nacional de videojuegos donde tomé la decisión final. Conocía a bastantes grupos y personas, pero fue el ver la cantidad de gente que se presentaba a estos concursos lo que me animó. Había muchísimas más personas desarrollando juegos de las que uno podía imaginar. Crear algo como Stratos se me hacía totalmente indispensable.

¿Cuál ha sido tu trabajo hasta la fecha en Stratos?
Je, je, je... pues todo. En los inicios, intentando convencer a la gente de que se uniese a la asociación, y hablando con desarrolladoras y distribuidoras para poder abrir caminos a los proyectos, y conseguir que nos dijesen algo cuando necesitasen gente. La página web, verdadero corazón de Stratos, ha sido posible gracias al esfuerzo de José Carlos García, *Carlitox*. Yo le echo una mano en contenidos pero gracias a él yo puedo dedicarme a otros asuntos: valoración de proyectos, búsqueda de comercialización, gestión de la base de datos de miembros, estrechar relaciones con empresas, indagar en el mercado... Lo bueno es ver que el esfuerzo pasado da sus frutos. Antes tenía que pelear muchísimo y ahora ya vienen directamente a Stratos porque somos un puntal importante en la industria del desarrollo.

¿Cuál es tu opinión personal acerca de la asociación?
Pues es como ver crecer a un hijo. Ves con ilusión cómo nace, cómo da sus primeros pasos, le dedicas todo el tiempo que puedes, sufres con él las malas rachas, disfrutas con él sus éxitos, y te sientes orgulloso cuando ves que está madurando y tu trabajo ha merecido la pena. En muchas ocasiones me han dado las gracias por haber creado Stratos y estar al pie del cañón día tras día, pero una de las frases que más me ha gustado ha sido la de "si Stratos no existiese habría que reinventarla. No abandonéis nunca". Ya sabes que, cuando estás en cualquier situación más o menos pública, siempre surgen las críticas. En unos casos son constructivas, y escucharlas ayudan a mejorar para el futuro. En otros casos son solamente ganas de... bueno, de fastidiar. Pero si evalúas los resultados, y te das cuenta de que hay gente agradecida por tu ayuda, te olvidas de los malos rollos.

Aparte de trabajar en Stratos, ¿tienes alguna otra relación con el mundo de la informática?
Por supuesto, tengo un trabajo. La asociación, al ser gratuita, solamente percibe un pequeño porcentaje de aquellas producciones que han dado frutos con su intervención, y se destina a cubrir los gastos que genera el mantenimiento: servidor web, teléfono, viajes, material... Y la mayoría de las veces ni siquiera

Cuadro2. Entrevista a Antonio Arteaga *(cont.)*

se cubren, por no añadir que absolutamente todo el tiempo que le dedicamos *Carlitox* y yo al asunto no está remunerado. Actualmente, estoy en un Gabinete de Comunicación, donde se trabaja en aspectos relacionados con Internet, desarrollos multimedia, soporte informático para las convenciones y asambleas, conversiones de soportes y formatos de material fotográfico, vídeo y CD-Rom, etc... La realización de mis propios juegos me aporta además dinero, y me permite dedicar algo de mi propio bolsillo para mantener la asociación cuando no hay ingresos mínimos.

¿Cuál crees que es el nivel de desarrollo actual en España?
Muy alto. Me estoy refiriendo al nivel que se puede observar en las grandes desarrolladoras. Los primeros puestos en las listas internacionales así lo demuestran, y no son casos aislados sino la punta de lanza de lo que aún tiene que venir. En nuestro país contamos con una ventaja que pocos países tienen: lo que no podemos alcanzar por falta de medios o inversión lo logramos a base de improvisación, tenacidad y, sobre todo, mucha creatividad. Si hay algo que distingue a los títulos españoles es, sin duda, su originalidad y jugabilidad. En algo se tiene que notar el carácter latino, ¿no?

¿Hay futuro para los programadores, grafistas y desarrolladores de videojuegos españoles?
¡Por supuesto! España ha tenido un agujero en el pasado, en el que los desarrolladores tenían que irse a otros países. Pero muchos ya han vuelto, porque la situación ha cambiado radicalmente. La cantera de desarrolladores es inmensa, y las posibilidades de adquirir experiencia son cada vez mayores. De acuerdo, aún no estamos al mismo nivel de retribución que en la media de las empresas extranjeras, pero muy pronto lo estaremos, porque los responsables de las empresas ya están apostando de cara al futuro.

¿Conoces DIV? ¿Qué opinas de dicha herramienta?
¿Quién no conoce DIV a estas alturas? Para serte sincero, al nivel de usuario muy poco (un par de juegos de prueba) porque provengo de la vieja escuela y mi línea de desarrollos siempre se ha basado en los lenguajes elementales: Basic, Pascal, C, C++... aunque con la proliferación de los distintos API's estamos llegando a tal nivel de abstracción que realmente el lenguaje de fondo no importa tanto. Ahora bien, DIV es una herramienta muy completa y, sobre todo, muy constructiva. Como principal virtud le encuentro el que aúne en un mismo entorno todos los apartados de la creación de un videojuego. Cuando se desarrolla un juego, por un lado están los programadores, que se dedican a picar y picar código, por otro los grafistas, que se curran la parte gráfica desde los sprites hasta las pantallas de menús y entorno, por otro los creadores de sonido y música, la figura del diseñador que debe encargarse de que todo tenga coherencia y orientarlo hacia el resultado final... En DIV lo tienes todo en un mismo sitio. Por supuesto, deben existir las personas mencionadas anteriormente, pero ya no se trata de piezas tan dispersas. DIV sirve, sobre todo, para que los que quieren formarse en el desarrollo de videojuegos adquieran unas nociones claras de lo que supone y se formen una base sólida para el futuro.

¿Conoces *DIVmanía*? ¿Qué opinas de la revista?
Hombre, estáis haciendo un reportaje sobre Stratos, tengo que opinar bien... No, ya en serio. Considero que es una revista necesaria. La cantidad de personas y grupos que desarrollan con DIV en nuestro país es enorme, y todo lo que se pueda aportar es siempre poco. Tanto en DIV como en cualquier otro entorno no sirve con tener la herramienta y lanzarse con los ojos cerrados a desarrollar. Surgen dudas y no vas a tener siempre a un gurú del desarrollo al lado a quien "brear" a preguntas. La revista cumple muy bien sus funciones, tanto por aportar soluciones y contenido como por servir como guía a la comunidad de desarrolladores de DIV. Ver lo que otras personas hacen sirve de aliciente.

¿Quieres añadir algo que no haya sido comentado en la entrevista?
Sí, me gustaría aprovechar la ocasión para alentar a todos los desarrolladores hispanos a que tomen conciencia de que no somos unos rebeldes peleando contra las normas establecidas por las multinacionales extranjeras, o de que solamente podemos hacer juegos y aplicaciones para nuestro entorno más cercano. Somos muchos, con muchas ganas de demostrar nuestra valía, y trabajando conjuntamente podemos lograr lo que de otra forma no sería posible. La unión hace la fuerza, la experiencia compartida acorta camino, y el futuro se abre a todos los que quieran formar parte de él. Si no sabéis cómo hacerlo, en Stratos estamos encantados de orientaros y ayudaros.



Air Forces **(Cuchipandi Technologies), arcade aéreo para Game Boy Color.**

Cuadro 3. Información adicional

Datos para contactar con Stratos:
Página web: *http://www.stratos-ad.com*

E-mail de los coordinadores:
Antonio Arteaga: *antonio@stratos-ad.com*
José Carlos García: *carlos@stratos-ad.com*
Teléfono: 925 26 81 25
IRC Hispano: *canal #programacion_stratos*

Casos prácticos de DIV2

Más utilidades

Como viene siendo habitual mes a mes, seguiremos con los trucos ya que, en el último número, quedó alguno en el tintero. Pero pasemos a la faena sin mas preámbulo, viendo truquillos y nuevas funciones de DIV2, uno a uno.

Demos un rápido repaso a todo lo que hemos visto hasta ahora. Espero que los trucos que hemos visto estos meses, así como los que veremos en este numero, os sean de utilidad.

Lo primero que vimos fueron algunos trucos sobre el entorno, y más específicamente, vimos el generador de sprites, el editor de mapas gráficos y el editor de mapas 3D. Luego, pasamos al lenguaje, algunas opciones de compilación, y distintas rutinas como las 3D,las rutinas sobre texto, las rutinas de ficheros, las funciones matemáticas y las rutinas del sonido y la música. En este último grupo nos quedamos, y este mes continuaremos justo por aquí. Así que, sin mas preámbulos, pasemos a ver las rutinas que quedan sobre sonido y música, pasando a estudiar una a una todas las funciones y algún que otro truco.

Vamos a ver las funciones nuevas relacionadas con la música y el sonido

Continuamos con el sonido y la música

Antes de comentar el último truco que nos queda en el tintero, y comentar las funciones nuevas relacionadas con el sonido y la música, recordemos los trucos que vimos el mes pasado, aunque sólo por encima. El primero de ellos era sobre los volúmenes, ya que DIV2 no tiene mucha potencia sonora en general, se comentó un truco para subir la potencia. El otro tenía que ver con los nuevos ficheros musicales, conocidos comúnmente como ficheros módulos, y los programas que existen para su realización.

El siguiente truco tiene que ver con una función que, aunque hay una parecida, realmente no existe, y es raro, ya que su utilidad es muy obvia. Se trata de la función *is_playing_sound()*, y también su homónima para canciones *is_playing_song()*. Con ellas podremos saber si un sonido o una canción ha acabado de ejecutarse. Es decir, si ha dejado de sonar y, por lo tanto, ha llegado al final del archivo de sonido. Su utilidad es muy clara, imaginar una intro, es decir, una serie de escenas donde haya una serie de personajes que hablen. Con estas funciones se podrá sincronizar toda la acción de la escena. Por ejemplo, hablará uno de los personajes, y el programador, mediante distintas instrucciones y código, hará que el personaje se mueva y haga el movimiento de los labios. Comprobando a cada paso si el sonido a dejado de sonar, en el caso de que ya haya finalizado, pasará al siguiente plano de la escena, reproduciendo otro sonido y esperando hasta que éste acabe.

Esta función también puede ser útil para otros menesteres. Imaginad un juego en el que el sonido sea totalmente esencial ya que, sin él, no se puede jugar ni utilizar el programa. Aunque hay funciones para detectar si existe una tarjeta de sonido, ésta puede estar utilizándose por otra aplicación, por lo que nuestro juego se quedaría sin sonido, con el problema que conlleva. Para solucionar esto, se puede grabar un sonido de tres o cuatros segundos de silencio, sin ningún ruido. Al comienzo del juego, se reproduciría el sonido, comprobando si está sonando con esta función, hasta que acabe. Esto se haría en un bucle de repetición, dando vueltas en él, hasta que no se haya reproducido totalmente el sonido. Por ultimo, se debe tener un temporizador al principio del programa, justo antes de ejecutar el sonido. Y una vez ejecutado, se comprobaría si realmente se ha reproducido, comparando el tiempo que ha pasado. Por ejemplo, imaginad un sonido con tres segundos de silencio. Si al finalizar de reproducirlo, no han pasado más de dos segundos, es que el sonido no ha sido reproducido y, por lo tanto, nuestro juego no tiene el control de la tarjeta de sonido. En este caso, ya podremos actuar en consecuencia, saliendo del programa o advirtiendo al usuario que no dispone de sonido accesible.



Una vez vistos los últimos trucos, pasaremos a ver todas las funciones nuevas relacionadas con la música y el sonido, como viene siendo habitual, daremos una pequeña descripción de cada una de ellas, para situarlas, debiendo consultar otra documentación, si se quiere ampliar la información.

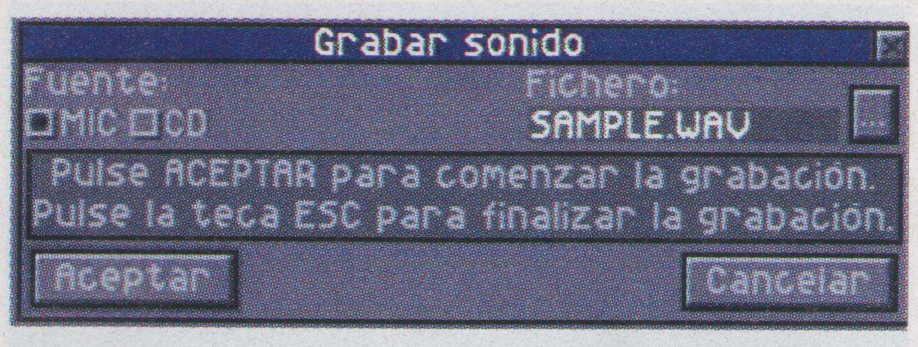
– *int load_song(nombremodulo,loop):* esta función sirve para cargar los módulos del tipo MOD, XM, IT, etc. en memoria. El primer parámetro es el nombre del fichero dentro del disco duro, y el segundo, es una bandera que indica si se quiere que la canción sea cíclica, es decir, que una vez llegado al final, empiece de nuevo. Devuelve un código, que es el que se debe utilizar luego para ejecutar la canción o descargarla de memoria.

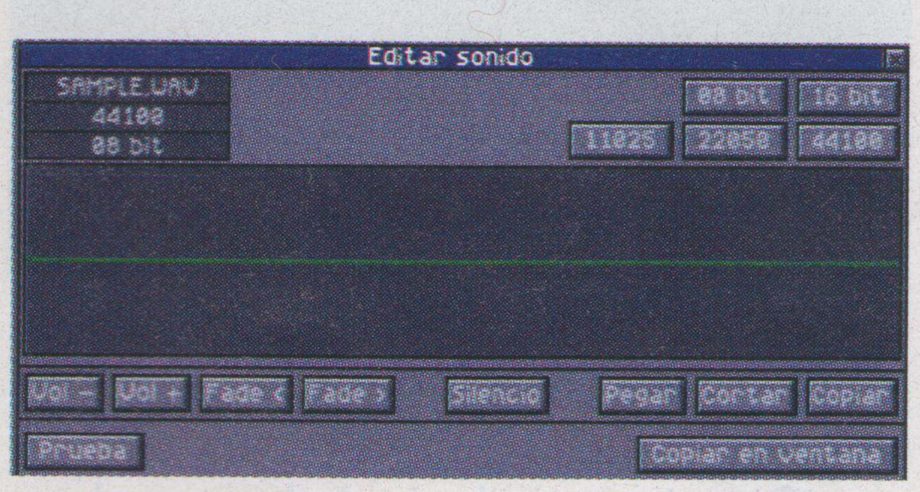
– *int unload_song(código):* descarga un fichero del tipo módulo, o canción con extensión MOD,XM, etc. de memoria. Como parámetro se debe indicar el código que se recogió cuando se cargó la canción en memoria.
– *int song(código):* toca una sola canción. El código que se debe pasar como parámetro es el que se recogió al cargar la canción en memoria. Se pueden tener varias canciones cargadas en memoria, pero únicamente se puede tocar una. No es buena idea tenerlas en memoria, ya que ocupan espacio y no cuesta nada cargarlas del disco duro cuando se vaya a tocar.
– *int stop_song():* ésta podría ser la función contraria a la anterior ya que detiene la canción que esté sonando en ese momento. Como únicamente puede estar una canción tocándose, no necesita ningún parámetro de código.
– *int set_song_pos(paterna):* las canciones del tipo módulo se dividen a su vez, en algo que se denomina comúnmente paternas, que son como trozos de canción que pueden usarse más de una vez dentro de la canción. Esta función pone la canción en una paterna determinada, ya que el parámetro que usa es el número de la paterna donde queremos situar a la canción.
– *int get_song_pos():* y continuando con las paternas, esta función devuelve la paterna que se está tocando en ese momento. Se podría decir que es la función contraria a la anterior, ya que si la anterior situaba, ésta devuelve la posición actual.
– *int get_song_line():* al igual que una canción del tipo módulo, se divide en lo que hemos denominado paternas, éstas a su vez también se dividen en líneas, teniendo normalmente 64 líneas cada paterna. Esta función devuelve la línea que se está tocando, así que con la función anterior y con ésta se puede saber exactamente en qué lugar se encuentra la canción. No es posible situar una canción en una línea determinada, únicamente es posible en una paterna, pero no en una línea, eso sí, podemos saber la posición exacta de la canción, que puede ser útil para algunas sincronizaciones con la imagen.
– *int is_playing_song():* y ya por último, en el tema de canciones tenemos esta función que nos dice si se está tocando un modulo de música del tipo MOD, S3M, XM, etc. Puede ser muy útil para sincronizaciones, ya que se puede esperar en el código a que la canción acabe para hacer determinadas acciones.
– *int is_playing_sound(canal):* esta función es la homónima pero para sonido del tipo .WAV o PCM, es decir, sampler de sonido. Como parámetro se debe indicar el canal que queremos testear ya que indica si se está tocando un sonido en dicho canal. Como se comentó, es muy útil para sincronizaciones, ya que se puede esperar, por ejemplo, a que un personaje acabe de hablar, para realizar animaciones e intros.

Primitivas gráficas

Una vez vistos todos los trucos de sonido y sus funciones asociadas, pasemos a ver las primitivas gráficas. Lo primero es definir una primitiva gráfica, que no es otra cosa que un objeto con un aspecto gráfico, por ejemplo, una línea, un cuadrado o un circulo que aparece en pantalla. Estas primitivas pueden ser útiles para muchas cosas, como veremos a continuación, con algunos trucos. También meteremos en este apartado algunas funciones que, aunque no son exactamente primitivas gráficas, sí tienen que ver con los gráficos, por lo que irán en este apartado. Pasemos a ver los trucos.

Como hemos comentado, las primitivas gráficas con cuadrados, círculos y líneas que se pueden pintar en pantalla. Un cuadrado o un círculo puede ser muy útil para seleccionar con el ratón, como ocurre en muchos juegos, sobre todo, de estrategia, donde se pueden seleccionar muchas unidades, pinchando un punto con el ratón, arrastrando y soltándolo, creando así un cuadrado, y seleccionando todo aquello que esté dentro de este cuadrado. De la misma manera que el cuadrado, también se puede hacer con un círculo.

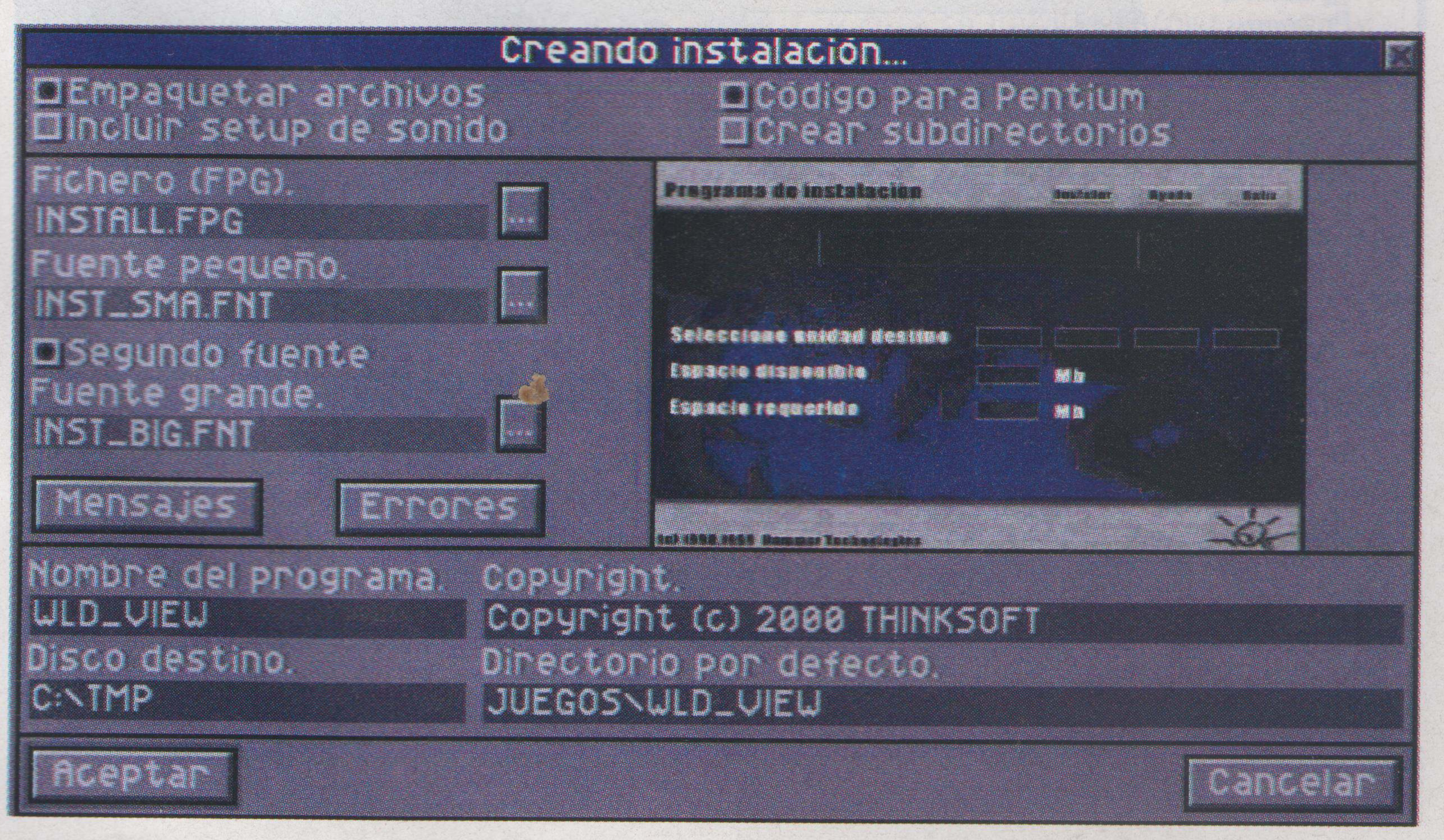




Y el círculo, o una serie de rayas, también puede ser muy útil para crear un punto de mira. Con las primitivas gráficas podremos crear desde la mirilla con la que dispararemos, hasta hacer una especie de disparos, a partir de líneas que salgan desde los extremos de la pantalla y se dirijan hacia nuestra mirilla. También se puede usar un gráfico "de los de siempre" como mirilla, y sólo utilizar las líneas para simular el rayo de disparo.

Las paternas son trozos de canción que pueden usarse más de una vez

Pasamos a otro tema gráfico, no es con primitivas, pero si es más útil. Este truco consiste en convertir un texto en un proceso. En la primera versión de DIV, para imprimir textos sólo disponíamos de dos funciones. Una de ellas escribía números en pantalla, o ,más exactamente, variables cuyo valor era un número, que se actualizaba automáticamente. Y la otra función escribía cadenas de caracteres, bien incluidas en una variable o directamente introducidas en la función entre comillas. Ahora disponemos de la función *write_in_map()*, que permite escribir un texto y convertirlo en un gráfico, y utilizarlo como cualquier otro gráfico que cargáramos en memoria. Al hacer esto, se puede asignar dicho mapa gráfico a un proceso, con las ventajas que conlleva, ya que podremos reescalarlo, moverlo, girarlo, utilizar transparencia, etc. Esto puede ser muy útil para realizar muchos efectos, y aunque antes podíamos tener en un gráfico algunos textos introducidos a mano desde el editor de mapas gráficos, éstos no eran modificables, y ahora sí lo son, ya





que con esta función podremos crearlos en tiempo de ejecución.

Otro truco relacionado con el trato de un elemento de manera distinta a la que veníamos haciendo hasta ahora, es el que veremos a continuación, que tiene que ver con los gráficos y la pantalla. Ahora, por ejemplo, se pueden hacer capturas mediante programación, además, también podemos grabar mapas en el disco duro e incluso crear nuevos mapas. Podremos, por ejemplo, hacer una captura de pantalla, y luego trabajar con ella, como si fuera otro gráfico más del fichero tipo fpg. Esto nos da unas grandes posibilidades a la hora de hacer fundidos de pantalla. Un caso práctico sería, el capturar la pantalla, para luego borrarla, poner nuestro gráfico creado, y redimensionarlo, con lo que conseguiríamos un efecto de fundido muy curioso.

Cada paterna consta normalmente de 64 líneas

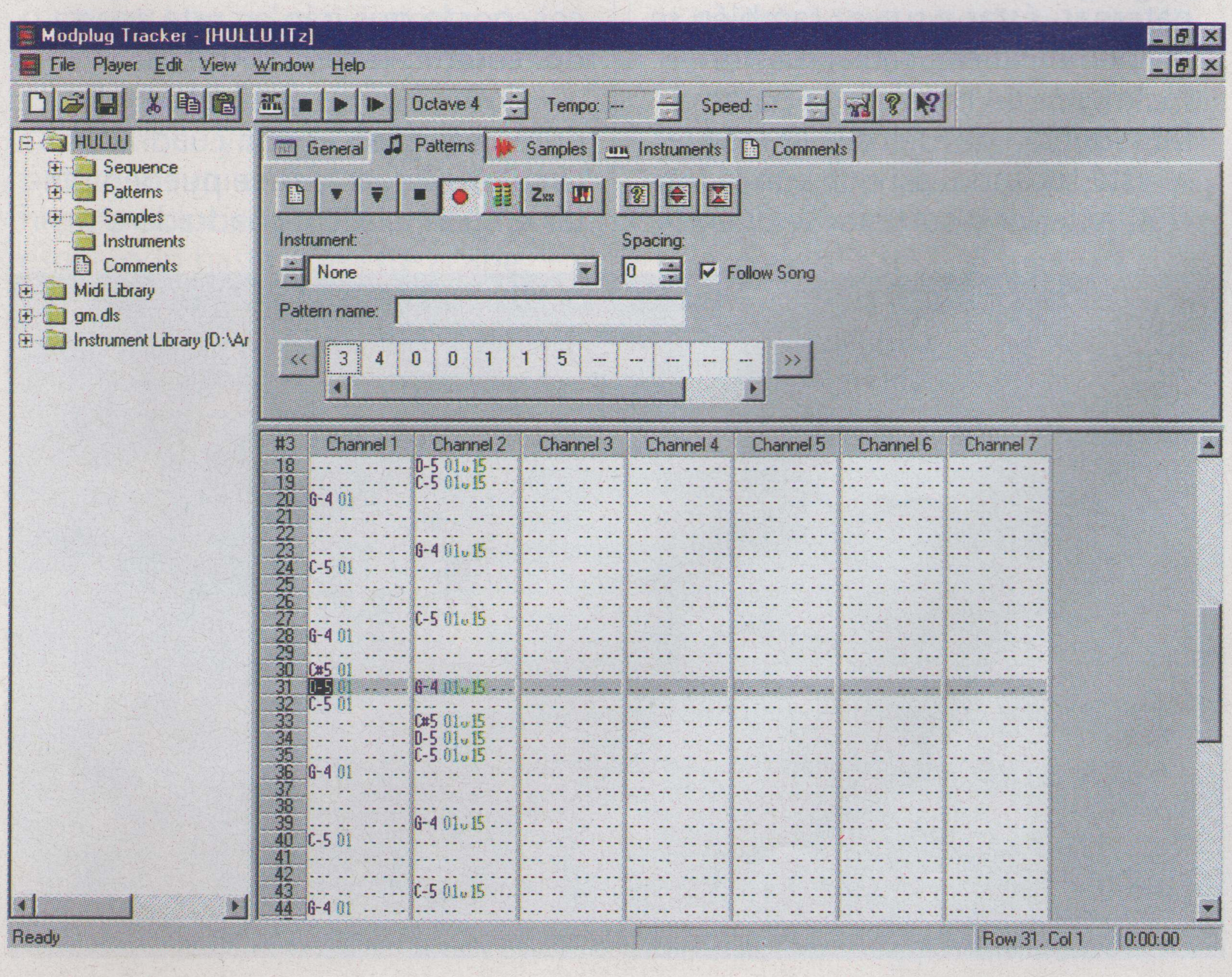
También podremos crear un gráfico desde cero, usando otras funciones de DIV para pintar puntos, o bloques gráficos. Esto, unido a la posibilidad de poder grabar cualquier gráfico en el disco, nos permite hacer capturas de pantallas o crear y grabar gráficos creados por el usuario. Además, se le puede asignar la pantalla, o un gráfico creado por nosotros a un proceso determinado, para luego tratarlo como tal, al igual que hacíamos con los textos, y reescalarlo, moverlo, girarlo, etc.

Con esto quedan vistos casi todos los trucos relacionados con gráficos. Hay que resaltar que la posibilidad de grabar gráficos en el disco, y crear nuevos gráficos en memoria para su posterior utilización es una gran mejora, ya que antes, al no disponer de estos medios el programador estaba limitado al fichero fpg o mapa que cargara desde disco, y no podía grabar nada en el disco, por lo que no tenía ningún dispositivo de salida. Ahora veremos las funciones relacionadas con primitivas gráficos, y creación y manipulación de gráficos creados mediante programación.

– *int draw(tipo,color,porcentaje, region,x0,y0,x1,y1):* esta función es la primera de las relacionadas con las primitivas. Con ella podremos dibujar cualquier tipo de primitiva, siendo éste el valor del primer parámetro. Los valores serán uno para línea recta, dos y tres para Rectángulo y Rectángulo relleno y, por último, tres y cuatro para utilizar una Elipse o Elipse rellena. El segundo parámetro que recibe la función es el color de dicha primitiva dentro de la paleta. Luego se debe indicar un porcentaje de transparencia de la primitiva, este valor podrá variar entre 0 y 15. También se indicará la región de pantalla donde aparecerá la primitiva, y la zona cuadrangular donde será pintada. Es decir, cualquier primitiva, tanto líneas, como elipses o rectángulos, se pueden definir mediante dos puntos, que son los últimos cuatro parámetros. La función devuelve un código, que identificará a la primitiva, para luego poder borrarla o mover
– *int move_draw(identificador, color, porcentaje, x0, y0, x1, y1):* con esta función podremos mover o cambiar el color, el tamaño o el porcentaje de transparencia. Para ello, debemos indicar como primer parámetro el identificador de la primitiva, además de los nuevos parámetros que deseemos.
– *int delete_draw(identificador):* y ésta es la última función dedicada a las primitivas, con ella borraremos tanto de la pantalla, como de la memoria, la primitiva que hemos creado. Para ello, deberemos indicar el identificador apropiado, devuelto por la rutina de creación.
– *int new_map(ancho,alto,centro x, centro y, color):* con esta función podremos crear un gráfico nuevo en memoria. Funciona de forma parecida a como lo hace la función *load_map()*, devolviendo un código identificador de mapa, similar al de esta función. Como parámetros, debemos indicar el ancho y alto del gráfico, así como las coordenadas del centro del gráfico y el color de fondo con el que se rellenará el mapa.
– *int screen_copy(región, fichero, grafico,x,y,ancho,alto):* con esta función conseguiremos copiar un trozo del contenido actual de una región de pantalla a un gráfico de un fichero determinado. Los parámetros son la región elegida, el código de fichero y gráfico donde se guardará el gráfico, y las coordenadas iniciales y tamaño del trozo de pantalla a copiar.
– *int save_map(fichero, grafico, nombre archivo):* si queremos hacer una captura de pantalla, con la función anterior y con ésta, podremos hacerlo, ya que permite grabar un gráfico en el disco duro. Tendremos que indicar el fichero y gráfico elegidos, así como el nombre del archivo dentro del disco duro.
– *int write_in_map(fuente, texto, centrado):* por último, con esta función podremos convertir un texto en un mapa gráfico. Los parámetros utilizados son el código de fuente, el texto elegido y el código de centrado, que es similar al usado con las funciones *write()* y *write_int()*. Devolverá un código de gráfico, similar al de las funciones *load_map()* y *new_map()*.

Funciones de memoria y sistema

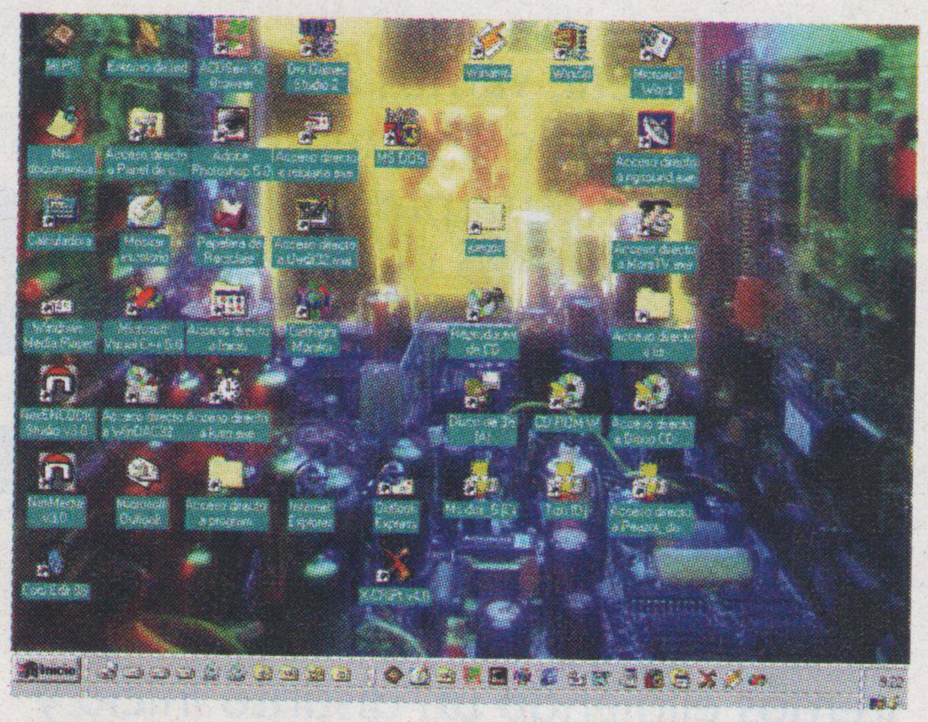
En este apartado veremos un par de funciones que, combinadas con otras, nos permiten describir un par de trucos. La primera función tiene que ver con la memoria, se trata de



la función *memory_free()*, que nos permite saber cuánta memoria hay libre, o lo que es lo mismo, memoria sin usar. Pero esta función por sí misma, no tiene ninguna utilidad, ya que el programador debe tener información sobre este dato. Existen numerosas formas de conocerlo, la primera es usando el debug de DIV, pulsando la tecla F12 durante el juego. También podemos imprimir en la pantalla este dato, para así visualizarlo, aunque el mejor método es hacer lo que denominamos comúnmente con un fichero log. Es decir, durante el transcurso del juego vamos comprobando la memoria libre, y el valor devuelto lo vamos grabando en un fichero en el disco, que luego podremos revisar tranquilamente, para comprobar si nuestro programa pierde memoria en alguna acción realizada.

Ahora pasemos al otro truco, del que ya vimos una versión, que se podría denominar ampliada. Se trata de usar la función *ignore_error()*. Dentro de las opciones de compilación, existe una que permite ignorar todos los errores que aparezcan en el juego. Esto puede ser hasta cierto punto peligroso, ya que puede producir ejecuciones del programa y efectos muy extraños, y no deseados. Si nuestro caso es el de que únicamente tenemos un error determinado dentro del programa, podremos usar la función *ignore_error()*, que únicamente ignora un error determinado, dejando que aparezcan los demás errores. Esto puede ser útil en determinados casos, aunque tanto la utilización de la opción de compilación, como de esta función, denota un estilo de programación un tanto pobre, no siendo recomendable su uso. Aunque siempre hay casos, en que es totalmente necesario, sobre todo, con los modos 8.

Otro aspecto, que aunque no tiene ninguna función relacionada, sí tiene que ver algo con el sistema es la creación de una nueva palabra reservada. Me explico, antes todos los procesos, eran eso, procesos, comenzaban con la palabra reservada *PROCESS*, debían tener normalmente un *FRAME*, para aparecer en pantalla, además de devolver el código identificador del proceso. También podíamos, retornar un valor propio pero, en este caso, no podíamos usar la palabra reservada *FRAME*, ya que dicho proceso no podía aparecer en pantalla. En ese caso, dicho proceso funcionaba, más que como un proceso, como una función. Ahora con DIV2, podremos indicar que el proceso es una función, para así clarificar de mejor manera los listados, usando la palabra reservada *FUNCTION*. Así, podremos separar, y tener claro qué parte de código es un proceso, y cuál es una función. Lógicamente, este tipo de funciones no podrá tener la palabra *FRAME*, además de retornar un valor. También podrá recibir parámetros, como cualquier otro proceso. Es decir, que la inclusión de esta palabra tiene un aspecto más clarificador, ya que en DIV1, ya se podía crear un proceso con el



mismo comportamiento, lo que pasa es que ahora podremos indicarlo de forma visible en el listado.

Ahora pasemos a ver las funciones relacionadas con este par de trucos, que son solamente dos.

– *int memory_free():* esta función devuelve un valor que nos permite saber la memoria libre que queda en el sistema.
– *int ignore_error(numero):* como parámetro de esta función deberemos indicar el número de error que queremos ignorar. Su uso es sencillo, pero como hemos comentando anteriormente, no es recomendable.

Funciones de color

Por último, pasamos a un apartado, que aunque no puede parecer importante, en algunos casos no será así. Antes de introducir los trucos que podemos conseguir trabajando con el color comentaremos un par de aspectos de DIV2, que podremos considerar como negativos.

La opción de "ignorar error" puede ser necesaria con los modos 8

El primero de ellos es el fallo de la función *roll_pallete()*. Esta función, que sirve para rotar la paleta, ha dejado de funcionar en DIV2. Este pequeño bug, es de carácter interno, por lo que no tiene solución, es decir, no se puede hacer nada para que dicha función vuelva a su estado normal. Aunque sí podemos conseguir un efecto parecido con las otras funciones. Aunque no podemos conocer el color que hay en una determinada posición, sí podemos designar una paleta establecida, que nosotros conozcamos totalmente, y moverla a mano, cambiando los valores, como veremos a continuación.

Ésa es la función que veremos ahora, ya que ahora existe la posibilidad de crear nuestra propia paleta desde cero. Esto, aunque parece muy simple, es muy potente, ya que al poder designar un color de la paleta con la función *set_color()*, podremos crear nuestra propia paleta, con todo lo que ello conlleva. Y uniendo esto al párrafo anterior, donde hablábamos de *roll_pallete()*, y la posibilidad de crear nuestro pro-

Cuadro 1. La lista de correo y el DIVCD

Aquí vamos a ver un par de temas relacionados con DIV. El primero es que la lista de correo ha cambiado de sitio. Si antes estaba en el servidor *onelist*, cuya página era *http://www.onelist.com*, ahora, han movido todas las listas a uno nuevo, llamado *egroups*, y cuya pagina es *http://www.egroups.com*. Por lo tanto, la lista pasaría a denominarse *canaldiv@egroups.com*.

Por cierto, últimamente la lista está muy revolucionada. Se tratan temas como la calidad de esta revista, ***DIVmanía***, el futuro de otra revista electrónica que muchos conocéis, y que se llama DIVnet, y otros temas. Pero un tema que ha resultado ser muy impactante es el siguiente. Una empresa, llamada Top Games, de origen argentino, ha sacado a la venta algunos juegos hechos con DIV, no se sabe a ciencia cierta si con el consentimiento de sus autores. En la lista, se ha optado por la siguiente solución: se ha decidido crear un CD, con todos los juegos de DIV posibles, cuyo nombre provisional es DIVCD. El proyecto está todavía en sus primeros pasos y esperamos que llegue a buen término. No es seguro pero, en principio, sería un CD con la mayor cantidad posible de juegos hechos con DIV gratuito, o lo más barato posible, y se distribuiría por distintas revistas o por alguna distribuidora.

Otro tema, también de interés es el DIVsite, un lugar en Internet que albergase todo el mundo de DIV. Esta pagina, o mejor podríamos llamarlo servidor, estaría creado y mantenido por los usuarios, y contendría revistas de electrónica, juegos, y todo lo que se ha generado, y lo que tiene que venir, sobre el mundo DIV. Desde estas líneas apoyamos dicha idea.

pio *roll_pallete*, vamos a ver un truco para conseguirlo. Podremos tener en una tabla, nuestra propia paleta preestablecida. Luego usando la función *set_color()*, la introduciremos dentro del programa, aunque sólo sea la parte que queremos rotar. Por último, para conseguir la rotación de colores, únicamente debemos cambiar el índice de la tabla. Es decir, iremos eligiendo un distinto inicio y fin, de selección de color, con esto conseguiremos que la paleta rote, como hacía la función *roll_pallete()*.

Además de poder crear una paleta única con todos los gráficos que tengamos, existe una nueva función en DIV2, cuya utilidad es muy obvia. Se trata de *force_pal()*, que nos permite tener una sola paleta para todo. Se acabó lo de convertir paletas y font de letras, de forma exterior a la programación. Ahora, usando esta función, mediante programación, conseguiremos que todos los gráficos y font, se adapten automáticamente a una paleta determinada. Esto nos quitará mucho trabajo, aunque debido al número de colores, mas de 16 millones, la adaptación no será siempre la deseada, simplemente, porque en la paleta que tengamos cargada no exista.

Por último, comentaremos una función cuya utilidad no ha encontrado el que escribe, aunque esto no signifique que no la tenga, simplemente, no se me ha dado ningún caso donde utilizarla. Se trata de *find_color()*, que se utiliza para buscar la posición de un color dentro de la paleta. Esta función, unida a *set_color()*, puede servir para cambiar dos colores determinados de posición.

Podemos tener en una tabla nuestra propia paleta de colores

Bueno, ahora pasemos a ver las funciones destinadas al color, de forma más detenida. La lista es la siguiente:

- *int set_color(color,r,g,b)*: esta función sirve para designar un color dentro de la paleta, es decir, crear un color propio. Como parámetros debemos indicar la posición de color dentro de la paleta, y los componentes RGB del color que queremos crear. Estos componentes RGB, son valores que definen el color. El componente R, que proviene del inglés, *Red*, designa la cantidad de color rojo que tendrá el que queremos crear. El componente G, *Green*, significa verde, y el componente B, *Blue*, significa azul. Juntando estos tres componentes, podremos conseguir cualquier color.
- *int find_color(r,g,b)*: esta función sirve para buscar un color determinado dentro de la paleta. Debemos utilizar como parámetros el color, dividido en componentes RGB, como hemos explicado en la función anterior. La función devuelve el color más cercano al indicado, y los componentes tendrán valores entre 0 y 63.
- *int force_pal(nombre archivo)*: por último, tenemos esta función que sirve para forzar al programa a utilizar solamente esta paleta. El entorno DIV convertirá automáticamente a dicha paleta todos los fondos o gráficos que carguemos en memoria. Debemos indicar como parámetro el nombre del archivo del cual usaremos la paleta. Para dejar de forzar la paleta, se usará como parámetro el valor 0, volviendo al funcionamiento normal. Es decir, los gráficos no se adaptarán a la paleta cargada en memoria, sino que utilizarán la propia.

Cuadro 2. Ediv y Fenix (El fenómeno DIV)

Se está creando por parte de usuarios otros programas para crear juegos. Éstos están basados en la esencia de DIV, se llaman Fenix y EDIV. Los dos entornos de programación son gratuitos, por lo que cualquier usuario puede descargarlos desde Internet para su posterior prueba. Aunque para la gente que no tiene conexión a la Red, desde estas líneas, se intentará hacer llegar estos productos a esta revista.

Y es que, realmente, DIV ha roto esquemas. El número de usuarios apuntados a sus listas de correo es enorme. En cualquier ida, si uno se pasa por el canal a eso de la media noche, se puede encontrar fácilmente a 10 ó 20 personas. El número de páginas que tratan sobre DIV es también enorme. Y esto es sólo la gente que tiene conexión a Internet, es decir, un tanto por ciento del número real de usuarios que, seguramente, será muy grande.

Por eso, la gente no quiere quedarse en DIV2 e intenta, por su cuenta, realizar productos parecidos, aunque sean gratuitos. Esto es así, porque la idea de DIV es genial, ya que la rapidez con que se puede hacer un videojuego con este producto es muy grande. Además de su sencillez de uso, y el que dentro de la misma herramienta se cuenta con utilidades de programación, de edición de gráficos, e incluso de manejo de sonido.

Bueno, sólo desear suerte a la gente que está trabajando en estos proyectos. Y, desde aquí esperamos que sus productos lleguen a buen fin. Consiguiendo, por lo tanto, terminarlos y darles salida, de la forma que sea, para el disfrute de todos los usuarios que así lo quieran.

Rutinas de red

Para empezar con esta sección habría que dejar muy claro que el que escribe no ha hecho pruebas con las rutinas de red, por lo cual, los consejos que se pueden dar son mínimos. Hasta aquí han llegado rumores de que tales rutinas funcionan, pero no se ha comprobado nada todavía. La mayor parte de la gente que lo ha intentado, ha tenido serios problemas para hacerlas funcionar. Es verdad, que dichas rutinas tienen muchos errores o bugs, y el número de pruebas que se hicieron con ellas antes de salir al mercado fue mínimo.

Esto pasa igual en el modo 8, que también queda por experimentar, aunque se han hecho muchas pruebas. Al contrario, las rutinas de red, más difíciles de probar, por el simple hecho de que hay que disponer de una red IPX y, por lo tanto, de más de un ordenador para poder hacer pruebas. Por eso, en modo 8, se han encontrado miles de trucos como, por ejemplo, no juntar más de dos sectores en un punto, intentar no unir paredes en la misma posición, no usar muchos vértices, etc. Sin embargo, esto no ocurre con las rutinas de red, donde las pruebas son mínimas y poco satisfactorias.

Por esta razón, para no crear un castillo en el aire explicando algo que realmente no se sabe a ciencia cierta, lo dejamos en este punto, pidiendo a cualquier lector, que si ha hecho pruebas satisfactorias con las rutinas de red, utilizando cualquier tipo de truco, se lo haga saber a la lista de DIV, a esta revista o, simplemente, al que escribe este artículo.

Despedida y cierre

Con este artículo se cierra la serie. Hemos ido recorriendo todas las novedades que incorpora DIV2, viendo algunos trucos sobre cada una de ellas. Espero que todos estos trucos os sean útiles o que, por lo menos, estos artículos os hayan servido para repasar todas las mejoras de DIV2. Ya sabéis que si tenéis alguna duda, siempre la podéis hacer llegar a la revista, o si lo preferís, a un servidor, cuya dirección de correo electrónico es: *tizo@100mbps.es*. Nada más, como cada mes, sólo queda decir lo siguiente: "Que os DIVirtáis programando".

Antonio Marchal (Tizo)

Mapeados Tiles en vista lateral

Infinidad de fases para tu juego de plataformas

Conectando con el anterior artículo de arcades y plataformas, dedicamos el espacio en esta ocasión a la creación e introducción al mundo de los tiles y los mapeados tiles en 2D de vista lateral. Un recurso que nos va a permitir crear escenarios diferentes en muy poco tiempo.

El mundo de los tiles siempre ha resultado una tediosa tarea y quizás un gran inconveniente para su propagación efectiva. Intentaremos pues, y debido a que muchos son los grafistas y programadores que desconocen lo que es este apasionante mundo, dar las notas iniciales para adentrarnos en él.

Los tiles han sido siempre un gran recurso en la programación de videojuegos, que nos permite crear grandes escenarios con fragmentos de lo que podríamos llamar un "mosaico gráfico".

Para la creación de un tile tenemos que saber el tipo de rejilla a utilizar

Un tile no es ni más ni menos que un pequeño fragmento de forma rectangular que, unido a una serie escasa de los mismos, tiene la virtud de crear escenarios de gran amplitud. Muestra de ello, son juegos como Super Mario World del que os enseñamos una muestra de sus tiles (ver foto 1). Lo que hace característico a un tile es su forma especialmente diseñada, como un patrón que puede ser ampliado hasta el infinito sin la sensación de repetición y su limitación se encuentra precisamente en la imaginación y astucia de su creador. Pero aún es mucho más que eso pues los años han demostrado que estos recursos tenían todavía mucho que ofrecer, y muestra de ello son los juegos de estrategia de última generación.

¿Cómo podemos empezar a crear tiles?

Para la creación de un tile, lo primero que tenemos que saber es el tipo de rejilla que vamos a utilizar. La rejilla (ver foto 2) será el espacio imaginario que nos servirá para encajar los tiles unos con otros y crear el escenario final. Es por ello imprescindible que todos los tiles sean, "en principio" del mismo tamaño. Sólo en principio porque aunque el tamaño real de los tiles sí sea efectivamente igual al del resto, con el uso de las transparencias (en DIV, el color 0 de la paleta cargada) podemos crear tiles no necesariamente cuadrados. Es también muy aconsejable, con vistas a la programación posterior que los tiles sean proporcionales a la resolución de la pantalla, es decir, a una resolución de 640 x 480 en la que sólo vamos a utilizar un scroll lateral en 2D (con lo que sólo debemos tener en cuenta los 640 píxeles de ancho de la pantalla) que sus tiles sean un divisor exacto del mismo, en este caso, anchos de 64 (64 x 10=640 píxeles) 32 (32 x 20=640 píxeles) o 16 (16 x 4=640 píxeles), con vistas a la programación de tileados con scroll automático que veremos en el próximo número.

El siguiente paso es imaginar el tile. Los tiles suelen representar fragmentos de objetos, como

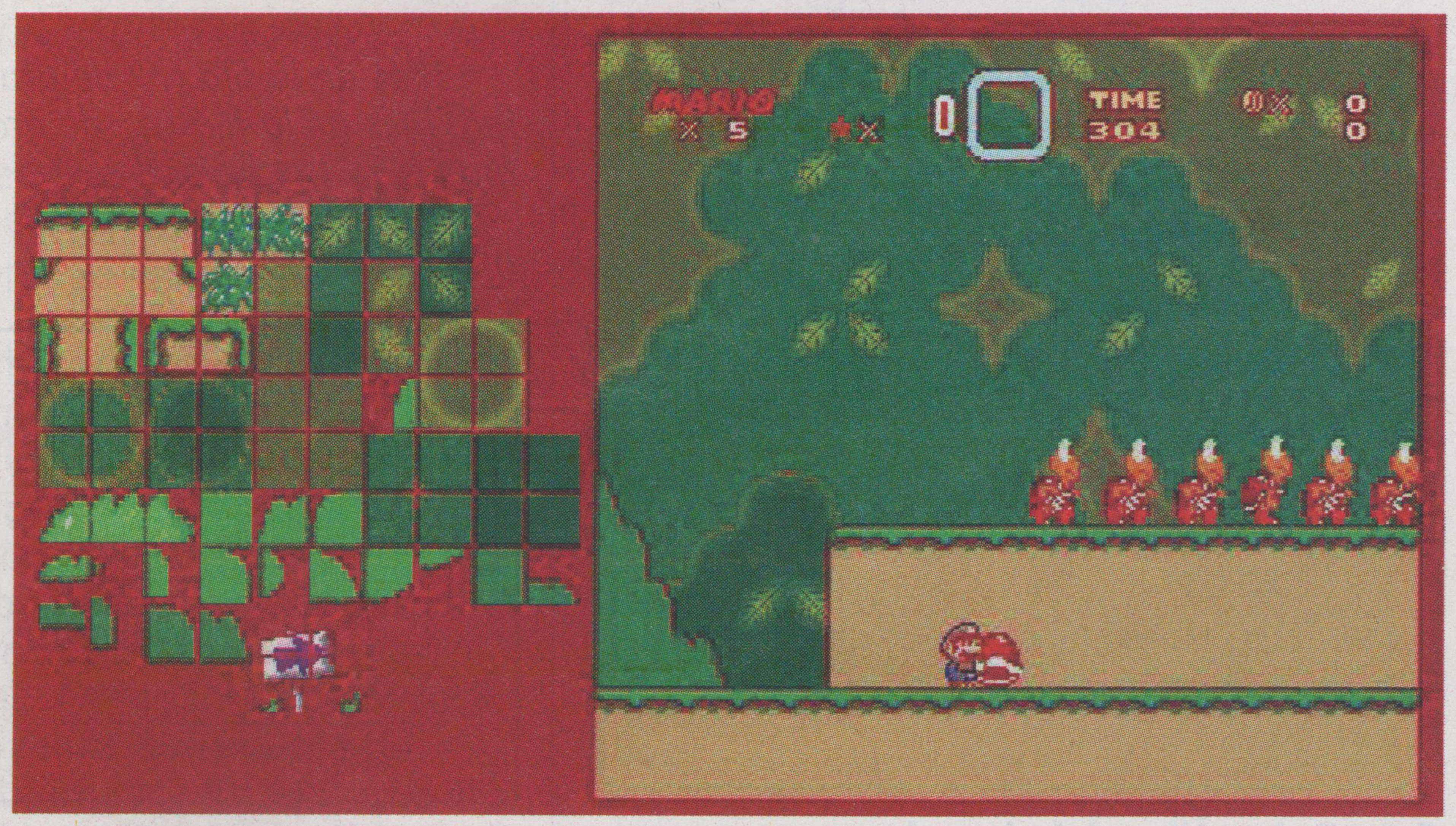


Foto 1. Los Tiles de Super Mario World.

pio *roll_pallete*, vamos a ver un truco para conseguirlo. Podremos tener en una tabla, nuestra propia paleta preestablecida. Luego usando la función *set_color()*, la introduciremos dentro del programa, aunque sólo sea la parte que queremos rotar. Por último, para conseguir la rotación de colores, únicamente debemos cambiar el índice de la tabla. Es decir, iremos eligiendo un distinto inicio y fin, de selección de color, con esto conseguiremos que la paleta rote, como hacía la función *roll_pallete()*.

Además de poder crear una paleta única con todos los gráficos que tengamos, existe una nueva función en DIV2, cuya utilidad es muy obvia. Se trata de *force_pal()*, que nos permite tener una sola paleta para todo. Se acabó lo de convertir paletas y font de letras, de forma exterior a la programación. Ahora, usando esta función, mediante programación, conseguiremos que todos los gráficos y font, se adapten automáticamente a una paleta determinada. Esto nos quitará mucho trabajo, aunque debido al número de colores, mas de 16 millones, la adaptación no será siempre la deseada, simplemente, porque en la paleta que tengamos cargada no exista.

Por último, comentaremos una función cuya utilidad no ha encontrado el que escribe, aunque esto no signifique que no la tenga, simplemente, no se me ha dado ningún caso donde utilizarla. Se trata de *find_color()*, que se utiliza para buscar la posición de un color dentro de la paleta. Esta función, unida a *set_color()*, puede servir para cambiar dos colores determinados de posición.

Podemos tener en una tabla nuestra propia paleta de colores

Bueno, ahora pasemos a ver las funciones destinadas al color, de forma más detenida. La lista es la siguiente:

– *int set_color(color,r,g,b):* esta función sirve para designar un color dentro de la paleta, es decir, crear un color propio. Como parámetros debemos indicar la posición de color dentro de la paleta, y los componentes RGB del color que queremos crear. Estos componentes RGB, son valores que definen el color. El componente R, que proviene del inglés, *Red*, designa la cantidad de color rojo que tendrá el que queremos crear. El componente G, *Green*, significa verde, y el componente B, *Blue*, significa azul. Juntando estos tres componentes, podremos conseguir cualquier color.
– *int find_color(r,g,b):* esta función sirve para buscar un color determinado dentro de la paleta. Debemos utilizar como parámetros el color, dividido en componentes RGB, como hemos explicado en la función anterior. La función devuelve el color más cercano al indicado, y los componentes tendrán valores entre 0 y 63.
– *int force_pal(nombre archivo):* por último, tenemos esta función que sirve para forzar al programa a utilizar solamente esta paleta. El entorno DIV convertirá automáticamente a dicha paleta todos los fondos o gráficos que carguemos en memoria. Debemos indicar como parámetro el nombre del archivo del cual usaremos la paleta. Para dejar de forzar la paleta, se usará como parámetro el valor 0, volviendo al funcionamiento normal. Es decir, los gráficos no se adaptarán a la paleta cargada en memoria, sino que utilizarán la propia.

Cuadro 2. Ediv y Fenix (El fenómeno DIV)

Se está creando por parte de usuarios otros programas para crear juegos. Éstos están basados en la esencia de DIV, se llaman Fenix y EDIV. Los dos entornos de programación son gratuitos, por lo que cualquier usuario puede descargarlos desde Internet para su posterior prueba. Aunque para la gente que no tiene conexión a la Red, desde estas líneas, se intentará hacer llegar estos productos a esta revista.

Y es que, realmente, DIV ha roto esquemas. El número de usuarios apuntados a sus listas de correo es enorme. En cualquier ida, si uno se pasa por el canal a eso de la media noche, se puede encontrar fácilmente a 10 ó 20 personas. El número de páginas que tratan sobre DIV es también enorme. Y esto es sólo la gente que tiene conexión a Internet, es decir, un tanto por ciento del número real de usuarios que, seguramente, será muy grande.

Por eso, la gente no quiere quedarse en DIV2 e intenta, por su cuenta, realizar productos parecidos, aunque sean gratuitos. Esto es así, porque la idea de DIV es genial, ya que la rapidez con que se puede hacer un videojuego con este producto es muy grande. Además de su sencillez de uso, y el que dentro de la misma herramienta se cuenta con utilidades de programación, de edición de gráficos, e incluso de manejo de sonido.

Bueno, sólo desear suerte a la gente que está trabajando en estos proyectos. Y, desde aquí esperamos que sus productos lleguen a buen fin. Consiguiendo, por lo tanto, terminarlos y darles salida, de la forma que sea, para el disfrute de todos los usuarios que así lo quieran.

Rutinas de red

Para empezar con esta sección habría que dejar muy claro que el que escribe no ha hecho pruebas con las rutinas de red, por lo cual, los consejos que se pueden dar son mínimos. Hasta aquí han llegado rumores de que tales rutinas funcionan, pero no se ha comprobado nada todavía. La mayor parte de la gente que lo ha intentado, ha tenido serios problemas para hacerlas funcionar. Es verdad, que dichas rutinas tienen muchos errores o bugs, y el número de pruebas que se hicieron con ellas antes de salir al mercado fue mínimo.

Esto pasa igual en el modo 8, que también queda por experimentar, aunque se han hecho muchas pruebas. Al contrario, las rutinas de red, más difíciles de probar, por el simple hecho de que hay que disponer de una red IPX y, por lo tanto, de más de un ordenador para poder hacer pruebas. Por eso, en modo 8, se han encontrado miles de trucos como, por ejemplo, no juntar más de dos sectores en un punto, intentar no unir paredes en la misma posición, no usar muchos vértices, etc. Sin embargo, esto no ocurre con las rutinas de red, donde las pruebas son mínimas y poco satisfactorias.

Por esta razón, para no crear un castillo en el aire explicando algo que realmente no se sabe a ciencia cierta, lo dejamos en este punto, pidiendo a cualquier lector, que si ha hecho pruebas satisfactorias con las rutinas de red, utilizando cualquier tipo de truco, se lo haga saber a la lista de DIV, a esta revista o, simplemente, al que escribe este artículo.

Despedida y cierre

Con este artículo se cierra la serie. Hemos ido recorriendo todas las novedades que incorpora DIV2, viendo algunos trucos sobre cada una de ellas. Espero que todos estos trucos os sean útiles o que, por lo menos, estos artículos os hayan servido para repasar todas las mejoras de DIV2. Ya sabéis que si tenéis alguna duda, siempre la podéis hacer llegar a la revista, o si lo preferís, a un servidor, cuya dirección de correo electrónico es: *tizo@100mbps.es*. Nada más, como cada mes, sólo queda decir lo siguiente: "Que os DIVirtáis programando".

Antonio Marchal (Tizo)

puede ser el terreno. Para crear un terreno no exclusivamente homogéneo podemos crearlo de forma sencilla en tres tiles, poniendo especial atención en el tile que va en el centro puesto que será el que vamos a multiplicar para crear espacios de terreno más o menos amplios.

Cuando fabricamos un tile, debemos tener en cuenta su posición en la pantalla y su función, más concretamente, cuáles de sus lados van a ser compartidos con otros tiles, que deberán tener una forma determinada para que conecten con el tile con el que ha de unirse. En el caso del terreno del que hablábamos antes, la parte final izquierda de la misma debe tener su lado derecho pensado de tal manera que la parte central pueda continuar sus mismas líneas o formas sin que se note que son tiles independientes y de igual forma si va a tener partes inferiores; a la parte central y la misma tarea para ambos lados. Y ésa es precisamente la tarea más complicada de los tiles, para la que hay que tener quizás un poco más de habilidad, especialmente indicado para aquellas personas que dominan el dibujo píxel a píxel. Pero nada mejor que la práctica para mejorar la técnica.

La rejilla nos indica qué tile va en cada posición y en cada momento

Recordad que aunque los tiles suelen ser pequeños, si deseáis hacer objetos más grandes podéis utilizar varios tiles para un solo objeto, para lo que no tienen que ser en absoluto cíclicos.

Programación

Pasemos a la práctica. En esta primera parte, de los dos artículos que explicarán con detalles la programación de los mapeados tiles en 2D con scroll lateral, vamos a aprender a traspasar un mapa creado a partir de tiles a una tabla de datos con el orden de los tiles y desde esa tabla de tiles al mapa del nuestro juego.

Hay que recordar en todo momento que entre las muchas virtudes de usar los mapeados tiles se encuentra la del ahorro de memoria y que, por tanto, vamos a intentar que nuestro juego use la memoria imprescindible, bien para dedicarla a otros menesteres (personajes y enemigos de gran tamaño, animaciones de los escenarios, o dotar al movimiento de una gran velocidad) o bien para que el programa pueda ser utilizado en ordenadores de gama baja.

Una vez tengamos los tiles preparados, creamos un mapa del tamaño que queramos para la fase y los encajamos creando el escenario en la forma que deseemos, dividimos el mapa con líneas según sean el tamaño de los tiles para conseguir una rejilla que nos permita saber qué tile va en cada posición en cada momento. Con esta rejilla, apuntamos los valores en una tabla bidimensional (en realidad unidimensional), anotando los que el tile de esa posición tenga otorgado en el archivo fpg donde estén almacenados, es decir, si el tile que vamos a pasar a la tabla tiene asignado el número 3 en el archivo fpg donde está guardado, entonces ése es el número que hay que insertar. Debería quedar como la que se muestra a continuación:

Tiles de la fila (nº de graph en fpg)Tiles de la fila (n_ de graph en fpg)
0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0, _ *fila 0*
0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0, _ *fila 1*
0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0, _ *fila 2*
1,2,2,2,2,3,0,0,1,2,2,
2,2,2,2,2,2,3,0, _ *fila 3*
5,4,4,4,4,6,0,0,5,4,4,4
,4,4,4,4,6,0, _ *fila 4*

Tabla que traspasaremos a DIV, como se señalaba, de forma unidimensional. En realidad es posible, incluso visualmente más correcto, dividir esta misma tabla de la siguiente forma, y no en forma lineal que sería de difícil visión.

```
Byte tiles[89] =
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  1,2,2,2,2,3,0,0,1,2,2,2,2,2,2,2,3,0,
  5,4,4,4,4,6,0,0,5,4,4,4,4,4,4,4,6,0;
//recuerde siempre el ";" al final.
```

Para acceder de forma bidimensional a esta tabla debemos encadenar dos bucles, uno para las filas y otros para las columnas o tiles de esa fila. Teniendo en cuenta que en este ejemplo cada fila contiene 18 tiles, para acceder a las filas posteriores a la 0, tendremos que multiplicar por 18 el número de la fila del tile a la que deseemos acceder, sumándole el número de la posición (o columna)del tile dentro de esa misma fila. Lo que traspasado a un bucle quedaría de la siguiente forma:

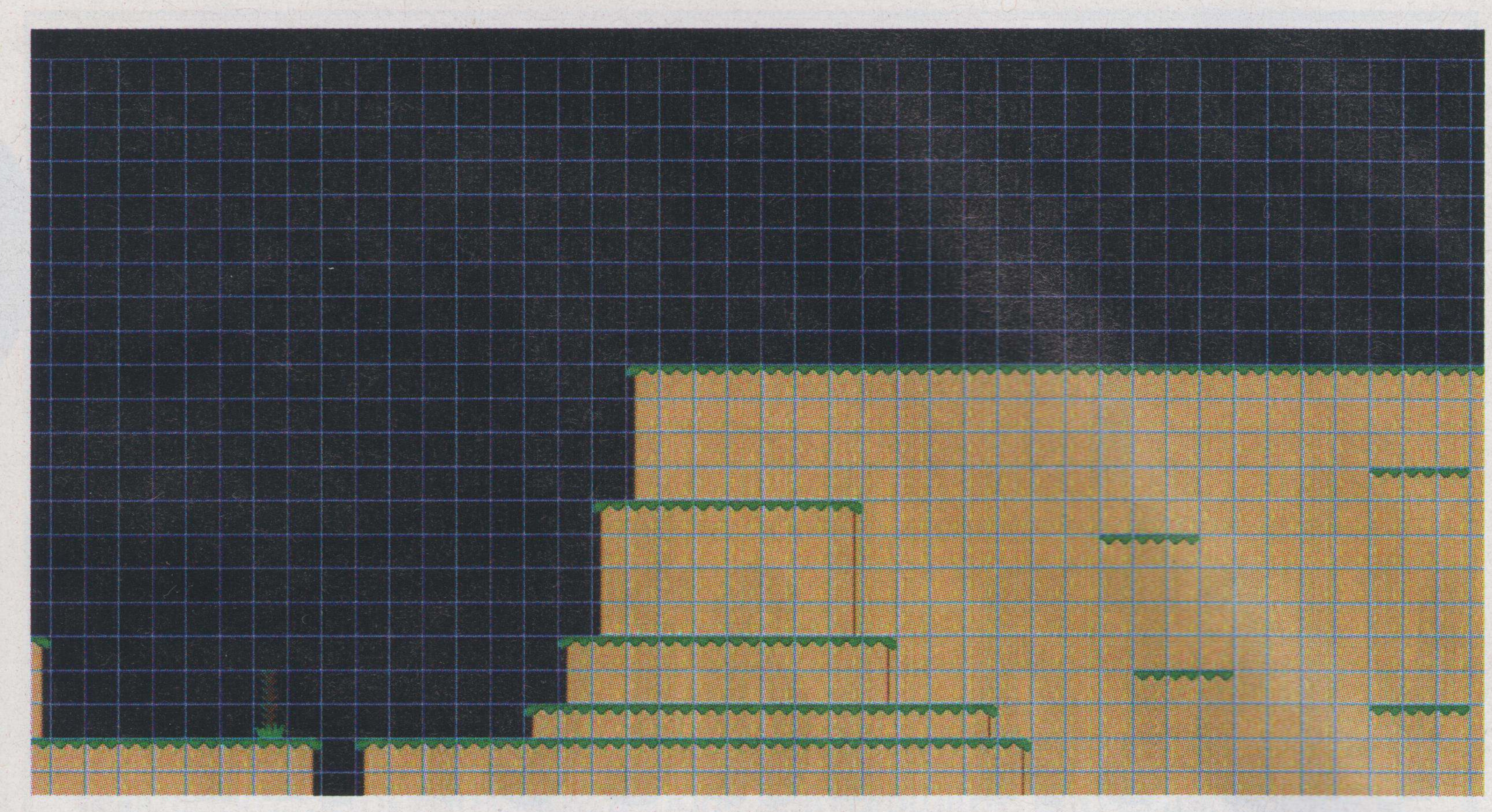


Foto 2. Los mapeados tiles se disponen en una rejilla imaginaria.

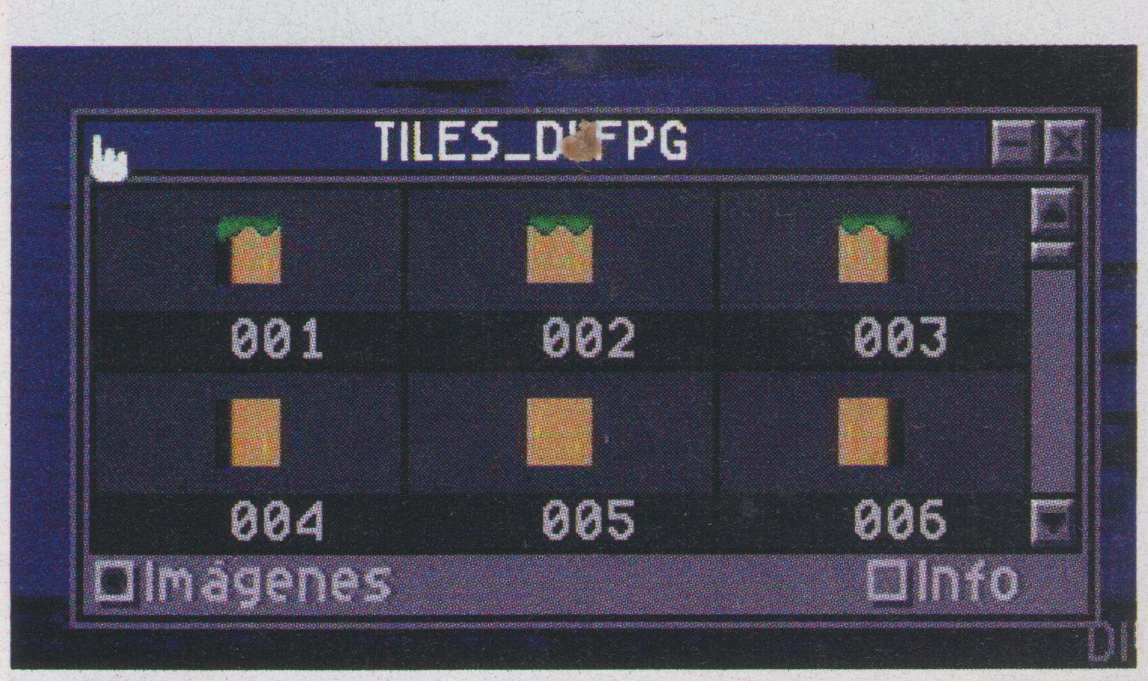


Foto 3. Los tiles del ejemplo.

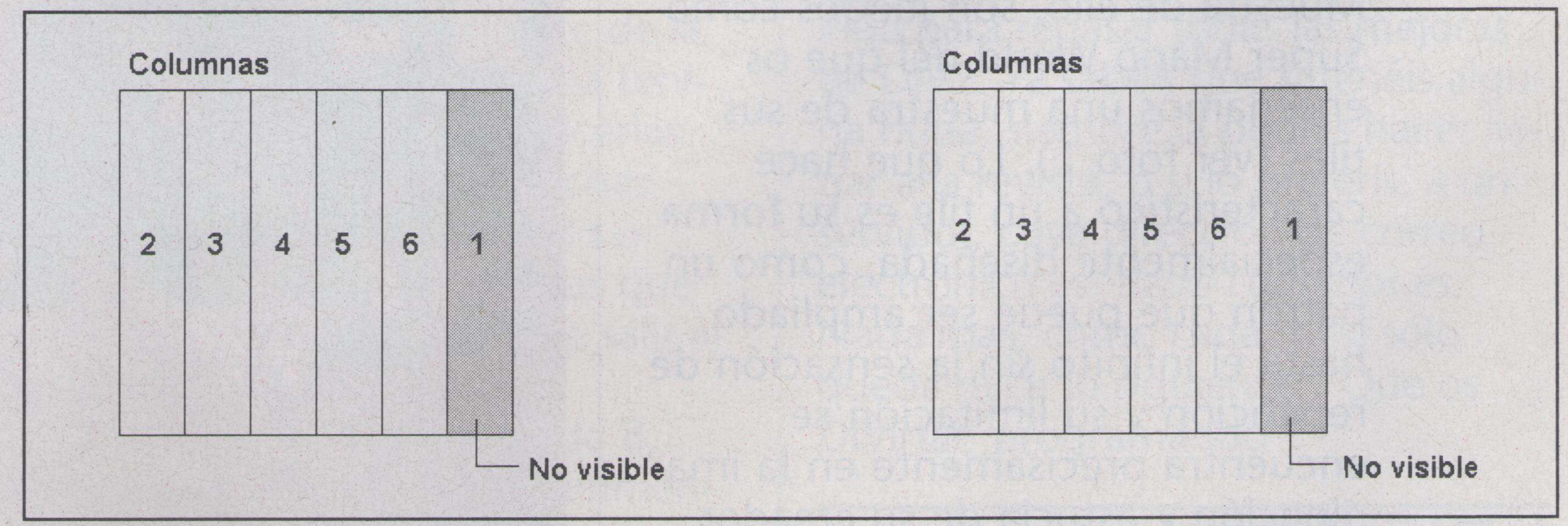


Foto 4. Figura 1 y Figura 2.

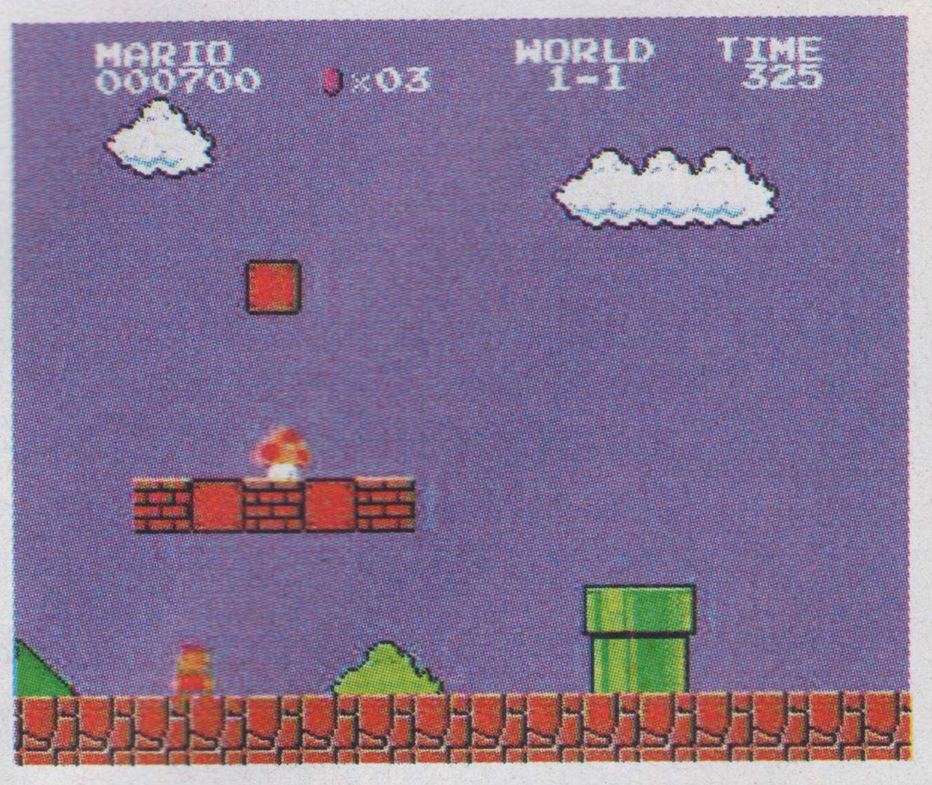


Foto 5. Super Mario Bros, todo un clásico de las plataformas.

```
From (fila=0 to 4)
From (columna= 0 to 17)
  [...]
End
End
```

Lo que conseguiremos con este bucle es un acceso lineal a los tiles. En cada fila, el bucle ejecuta los 18 pasos (0-17) para la lectura de las respectivas columnas correspondientes a tiles de la fila en cuestión. Por lo que sólo nos queda acceder a los tiles y encajarlos en nuestro mapa con *map_put* que nos permite introducir un gráfico en otro. Tan sólo un problema, recordad que *map_put* encaja los gráficos dentro de otros pero siempre dentro del mismo fichero fpg. Esto quiere decir que si creáis el mapa con *new_map*, no os será posible encajar tiles que no sean correspondientes al fichero 0, que es el fichero por defecto de los mapas creados en tiempo de ejecución. Teniendo en cuenta esto, sólo no quedan dos soluciones:

A) Que el fichero 0 que carguemos siempre sea el correspondientes a los tiles.
B) Que dentro de cada fichero incluyamos un mapa del tamaño de la fase de color transparente.

Veamos, ahora sí, cómo sería la función que nos permitirá encajar los tiles en el mapa:

```
From (fila=0 to 4)
From (columna=0 to 17)

  I= tiles[(fila*18)+columna];
  If (i>0)

     Map_put(0, nº_map_destino,tiles[(fila*18)+columna],
         columna*acho_tile,fila*alto_tile);
     End
  End
  End
```

Como se puede observar, en el bucle accedemos al *n* de gráfico que devuelve la posición del tile

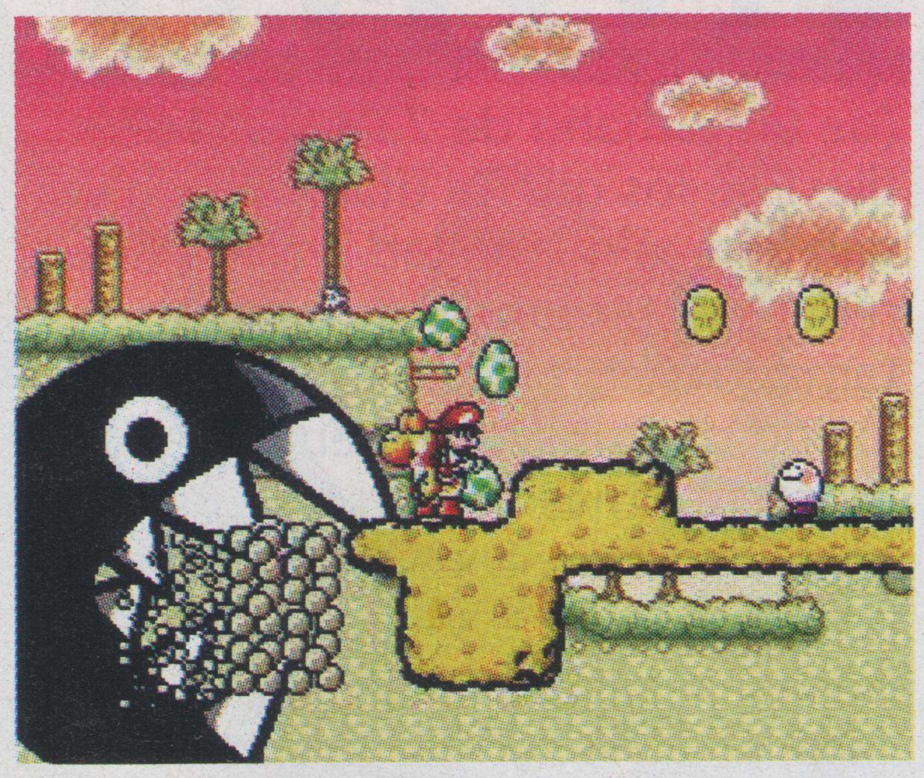


Foto 6. El género de las plataformas cada vez más complejo.

consultado y sólo accedemos a introducirlo en el mapa si el *graph* devuelto no es 0. Igualmente, para situarlos en el mapa, utilizamos una sencilla fórmula que consiste en multiplicar la columna por el ancho del tile, que nos dará su posición horizontal, y la fila por el alto del tile, que nos dará la posición vertical del mismo. Con esto, ya sólo nos queda crear el scroll correspondiente para mover horizontalmente el mapa, en caso de que sea mayor que la resolución que deseemos.

Para realizar un scroll necesitamos al menos un mapa, que será el de la fase, aunque se pueden poner algún otro que haga de fondo para crear un scroll parallax que podría dar un toque de calidad a nuestro juego, y también un proceso, en este caso el principal que activaremos como la scroll camera, con *scroll.camera=id.* Quedando algo así:

```
START_SCROLL(0,fpg_tiles,
9,0,0,0);
// en este caso dejamos en blanco el segundo mapa y no permitimos que el mapa se repita de forma cíclica
Scroll.camera=id;
```

Y para controlar el scroll simplemente:

```
If (key(_left))
  If (x>0) x-=4; END
End
If (key(_right))
  If (x<1500) x+= 4 END
END
```

Conceptos básicos de un mapper

Un mapper es un programa que permite hacer de forma automática las tablas de tiles a partir de un diseño que nosotros mismos hayamos definido para un determinado escenario. Es bastante útil y práctico porque realizar estas actividades "a mano" puede resultar bastante monótono. Las utilidades más características de los mismos incluyen una rejilla que adapta de forma automática los tiles que el usuario ha seleccionado para una casilla determinada de esa rejilla. Para evitar que el usuario coloque un tile en una zona que no es exactamente la correcta podemos mirar las coordenadas del ratón en el momento de la pulsación y colocarlo en el cuadrante que corresponda a esa situación del ratón. Por ejemplo, si se trata de una rejilla de tiles de 20 x 20 podemos comprobar en qué cuadrante está el ratón dividiendo por esos mismos valores las coordenadas de x e y: *x/20=columna* y *y/20=fila.* Otro aspecto importante es la posibilidad de automatizar totalmente la creación de escenarios, es decir, no sólo presentar los tiles la parte derecha de la pantalla para que el usuario los pulse y los ponga en la rejilla sino poner tan sólo iconos y usar el método de pulsar y arrastrar el ratón para que, por ejemplo, una sección de terreno se ponga de forma automática correctamente, es decir, de todas sus partes (parte izquierda, central y derecha, terreno debajo de terreno, etc) con lo que podremos incluso dar la posibilidad al jugador de crear él mismo sus propias fases sin ningún tipo de problema. En el próximo número se mostrará un sencillo ejemplo de cómo crear un mapeador de tiles y comentaremos las partes más importantes del código.

Un mapper es un programa que permite hacer tablas de tiles

Un mapeado tile con scroll avanzado con scroll automático

Decíamos anteriormente que una de las grandes virtudes que tienen los mapeados tiles es su bajo uso de los recursos del sistema. Este no es el caso del ejemplo que damos en este número pues un mapa de 1500 x 480 píxeles puede pesar bastante en memoria. Sólo es un ejemplo para mostrar cómo se empiezan a manejar los mapeados tiles y su programación. Una vez terminado este paso, debemos idear un sistema que de



Foto 7. Los Tiles, un recurso acogido en mucho géneros de videojuegos.

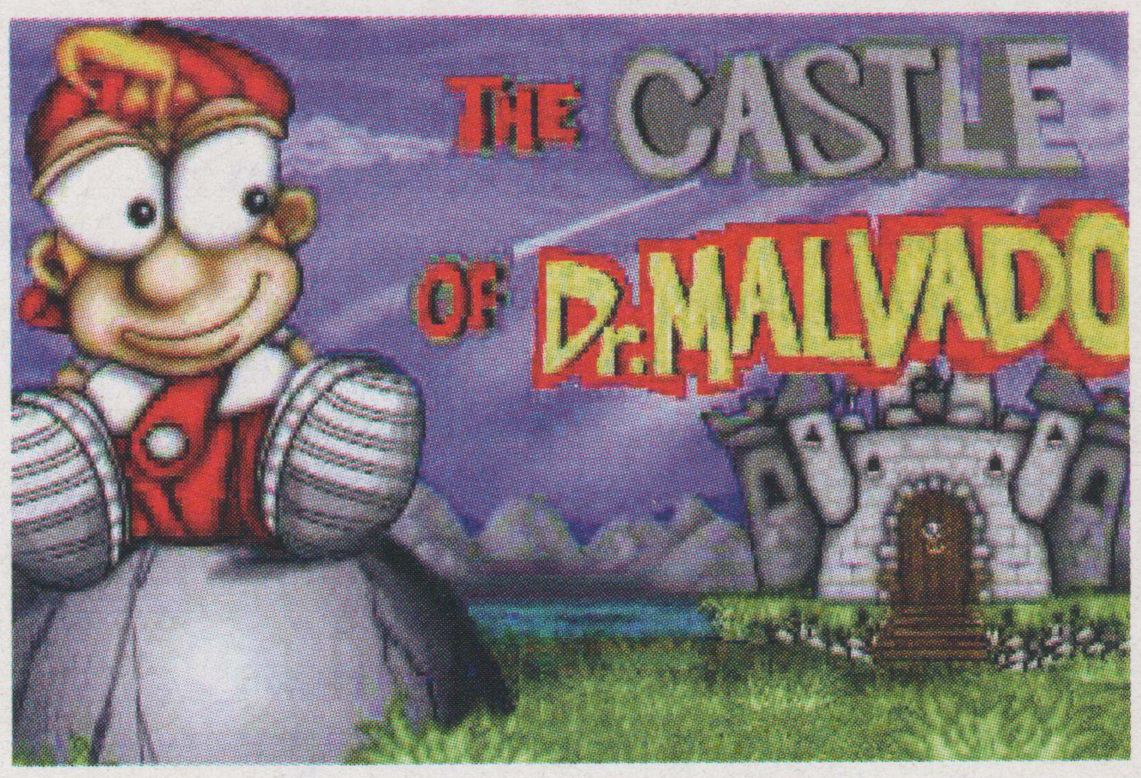


Foto 8. El inolvidable Dr. Malvado, el mejor plataformas de DIV.

forma automática muestre en la pantalla sólo lo que estamos viendo exclusivamente. Pensemos por un momento en los programas de estrategia que en su gran mayoría usan igualmente técnicas de mapeado de tiles. Si vemos el caso de Warcraft, las fases en extensión suelen ser bastante amplias superando los mapas incluso los 5000 x 5000 píxeles e inumerables personajes interactuando de forma continua en tiempo real. ¿Cómo es posible algo semejante de forma fluida? Pues la única respuesta es no mostrando lo que no se ve en ese momento, es decir, mostrar sólo aquellos gráficos que exclusivamente se están visualizando, suspendiendo la parte gráfica de los mismos que no lo están. Algo que traducido a los mapas de tiles se traduce en mostrar sólo aquellas fracciones de tiles que se están visualizando e ir actualizando la parte correspondiente a las regiones que van a ser vistas inmediatamente. En DIV, el único método posible que permite semejante cosa es la interacción entre un scroll y *new_map* de la forma que se va a describir teóricamente a continuación y prácticamente con un ejemplo en el siguiente número de acción y plataformas.

Los tiles suelen ser pequeños, para un objeto grande se usan varios tiles

Teniendo como base la misma tabla que usamos para el ejemplo que viene con el CD de la revista (de un tamaño de 73 x 23=1679 tiles) de los que se utilizan en pantalla tan sólo la mitad, vamos a reducir el tamaño del mapa al tamaño de un mapa igual a la resolución que utilicemos en ese momento más un incremento en su ancho y alto correspondiente al tamaño también de ancho y alto de tipo de tiles que usemos, en este caso de 20 x 20, por lo que en una resolución de 640 x 480 deberíamos utilizar un mapa de 660 x 500 y lo ponemos como mapa principal de un scroll cíclico (horizontalmente, o también verticalmente si vamos a esa posibilidad). Procedemos de la misma forma que se explicó y ponemos los tiles correspondientes a toda la pantalla de visión y la parte no visible de actualización que le añadimos. Precisamente en esa región de 20 píxeles que no se ve, será la parte que iremos actualizando según vayamos necesitándola, esto es, que cuando el jugador mueva el mapa hacia la derecha, el scroll moverá las regiones visibles quedando no visibles las primeras que aparecieron y entrando en las regiones visibles la parte que nosotros dejábamos para actualizar. Veámoslo de forma gráfica con un ejemplo.

Imaginemos una rejilla de tan sólo seis columnas (ver fig.1), donde la última no es visible y para poder actualizar los tiles sin que el usuario vea ningún cambio. Cuando el jugador mueve el mapa y la región 6 que antes no era visible se muestra en su totalidad en la pantalla, la región 1 queda totalmente oculta en la parte que ocupaba la 6, por lo que nuestro algoritmo deberá actualizar aquellas zonas del mapa que no estén visibles en ese momento.



Ejemplo de juego con scroll lateral.

Otra cuestión muy importante a tener en cuenta a la hora de crear un scroll automático de tiles avanzado es los mapas de durezas necesarios para que los personajes y enemigos puedan adaptarse a los mapas creados. Debido a que sería igualmente un inconveniente, en lo que a requerimiento de memoria supondría, tener cargado en memoria un mapa sólo para medir las durezas de nuestro mapeado de tiles. Para resolver esta cuestión, lo único que debemos hacer es un tileado paralelo al nuestro que, incluso aprovechando la misma rejilla y con los mismos números que anteriormente tenían los tiles, crear copias de esos mismos tiles que usábamos, pasarlo a escala de grises y pintar sobre ellos las zonas que impiden pasar o que permiten que el personaje pueda pasar por encima. En aquellos tiles que sean indiferentes para el mapa de durezas, es decir, que no tienen ningún efecto con los personajes o enemigos, simplemente, podemos dejarlos como transparentes.

Dejaremos el resto para el siguiente número que lo dedicaremos en exclusiva a comentar el código del ejemplo, en el que se mostrará un ejemplo de scroll tile avanzado, un mapper para mostrar cómo automatizar las tareas. Mucha teoría y muy poca práctica en este número, en el siguiente tendremos todo lo contrario. Hasta entonces podéis mandar cualquier tipo de duda o sugerencia en relación con la sección a *migens@teleline.es* o en la lista del canal en egroups (*www.egroups.com*) en *canaldiv@egroups.com*.

José Antonio Migens Gómez (ViTAL).
migens@teleline.es

Dudas de los lectores

Nos llega la consulta de un usuario preocupado por un problema que ha encontrado con DIV, provocado por el intérprete de la sintaxis de DIV que se queda colgado cuando se declara cualquier tipo de variable con el único texto "con". No aparece el problema si está acompañado de cualquier otro tipo de carácter.

Respuesta

Curioso problema el que nos comentas. Como al parecer el problema sólo ocurren con las variables strings con el texto "con" para utilizarlo en las comparaciones en cualquier texto o cualquier otro tipo de acción podemos añadirle en la declaración de la variable cualquier tipo de carácter al final y dentro del programa, al inicio o en el mismo momento que te haga falta eliminar el último carácter de la variable. Esto es posible debido a que parece que el problema se encuentra en el momento de interpretar la validez del código y no en el tiempo de ejecución.


Millennium
Chess
Un juego ancestral de estrategia
El mejor ajedrez virtual
Emula a los maestros del tablero
Desarrolla tu inteligencia
Perfecciona tu estilo ajedrecístico
Sólo 18,00 €
PC CD rom 2.995 Ptas
De venta en quioscos, grandes superficies y tiendas especializadas
Disfruta del milenario juego del ajedrez desde otra dimensión. "Millennium Chess" es un simulador de ajedrez con el que podrás desafiar a familiares y amigos o al propio ordenador. Incluye manual con amplia información
PC CD rom
COMPATIBLE WINDOWS 98
Software en castellano
Digital Dreams Multimedia
Digital Dreams Multimedia
C/ Alfonso Gómez, 42, nave 1-1-2
28037 Madrid (España)
Fax: +34 91 304 17 97
www.ddmultimedia.com
Para más información, llamar al teléfono +34 91 304 06 22

Trabajando con cadenas

Usando cadenas de texto

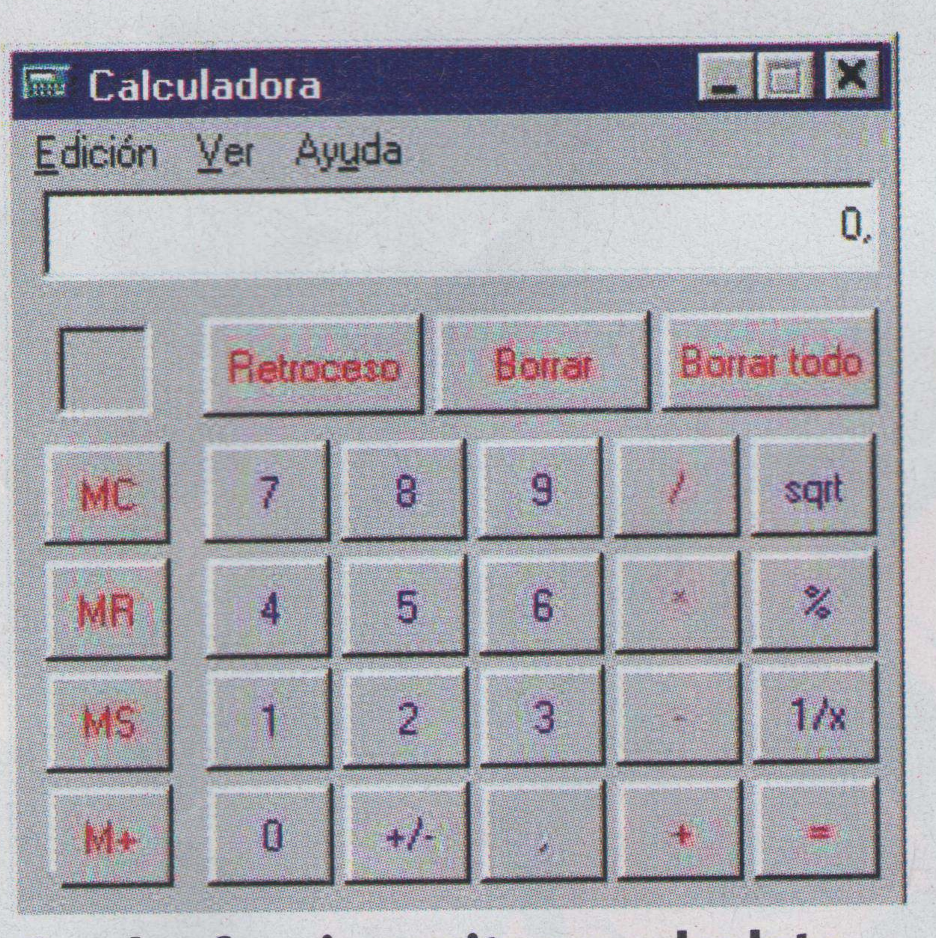
Un aspecto muy importante de un juego es la interactividad con el usuario. Es por ello que conviene diversificar las formas de comunicación con el jugador. Encontraremos en las cadenas una herramienta muy útil para construir interfaces hombre-máquina más eficaces.

Hasta ahora, tan sólo nos hemos comunicado con nuestros juegos mediante el teclado, joystick y ratón, para realizar desplazamientos de procesos. Pero en muchas ocasiones, desearemos utilizar el teclado para escribir texto y poder procesarlo posteriormente. Un claro ejemplo es una tabla de récords, en la que debemos introducir nuestras iniciales. En este número veremos cómo podemos captar y operar con cadenas de texto, pero para ello, en primer lugar, debemos conocer la nueva estructura de declaración de datos en DIV 2: los tipos de datos.

Tipos de datos

¿Qué es un tipo de datos? No es más que la forma de almacenamiento de los distintos valores de un programa. En la primera versión de DIV, podíamos declarar variables globales, locales y privadas, sin atender al tipo de dato que representaban, ya que todos se almacenaban de la misma forma. Esto también se puede realizar en DIV 2, si bien es recomendable utilizar la nueva capacidad de que dispone. Con los tipos de datos, podemos indicar la forma de almacenarlos en nuestro juego. Muchos de vosotros pensareis que es una carga más a la hora de programar. Es cierto, pero nos proporciona una mayor robustez en el código producido, ya que se pueden detectar con mayor facilidad errores como la asignación de un valor de cadena a una variable numérica.

Un tipo de datos es la forma de almacenamiento de los valores de un programa



Con las funciones itoa y calculate podemos usar DIV 2 como si de una calculadora se tratara.

En DIV 2 disponemos de varios tipos de datos, que describimos a continuación:

- **INT:** es el valor por defecto, es decir, el que se toma si no se indica ningún tipo de dato. Era el usado por la primera versión de DIV para todas sus variables numéricas. Su rango de valores oscila entre –2147483648 y +2147483647.
- **WORD:** la variable tiene un rango de valores entre 65535 y 0. No admite números negativos, con lo que para usarlos, debemos utilizar el tipo de datos INT.
- **BYTE:** toma datos entre 255 y 0.
- **STRING:** es un tipo específico de tablas que nos permitirá almacenar una cadena de caracteres, es decir, una sucesión de letras.

Todas las funciones del lenguaje han sido modificadas para aceptar como parámetros estos nuevos tipos de datos, si bien, se sigue manteniendo compatibilidad al 100% con la versión anterior.

Además de estos tipos de datos, disponemos de otras estructuras que nos permiten acceder a los datos de una forma mucho más cómoda y segura. Estas son las tablas y las estructuras.

Las tablas no son más que una lista de valores a los que se puede acceder mediante su posición en la misma, es decir, si tenemos una lista de valores BYTE como la que sigue:

2,4,6,8,7,1,2,10

Para acceder al valor 6, debemos indicar su posición, en este caso la posición 2 (comenzando por 0). Gráficamente, una tabla puede representarse de la siguiente forma:

Posición	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*
Valor	*2*	*4*	*6*	*8*	*7*	*1*	*2*	*10*

Para crear una tabla debemos seguir la siguiente sintaxis:

<tipo de dato> nombre tabla [tamaño de la tabla];

Para asignar a cada posición un dato (siempre del tipo de dato indicado en la declaración), debemos seguir la siguiente sintaxis:

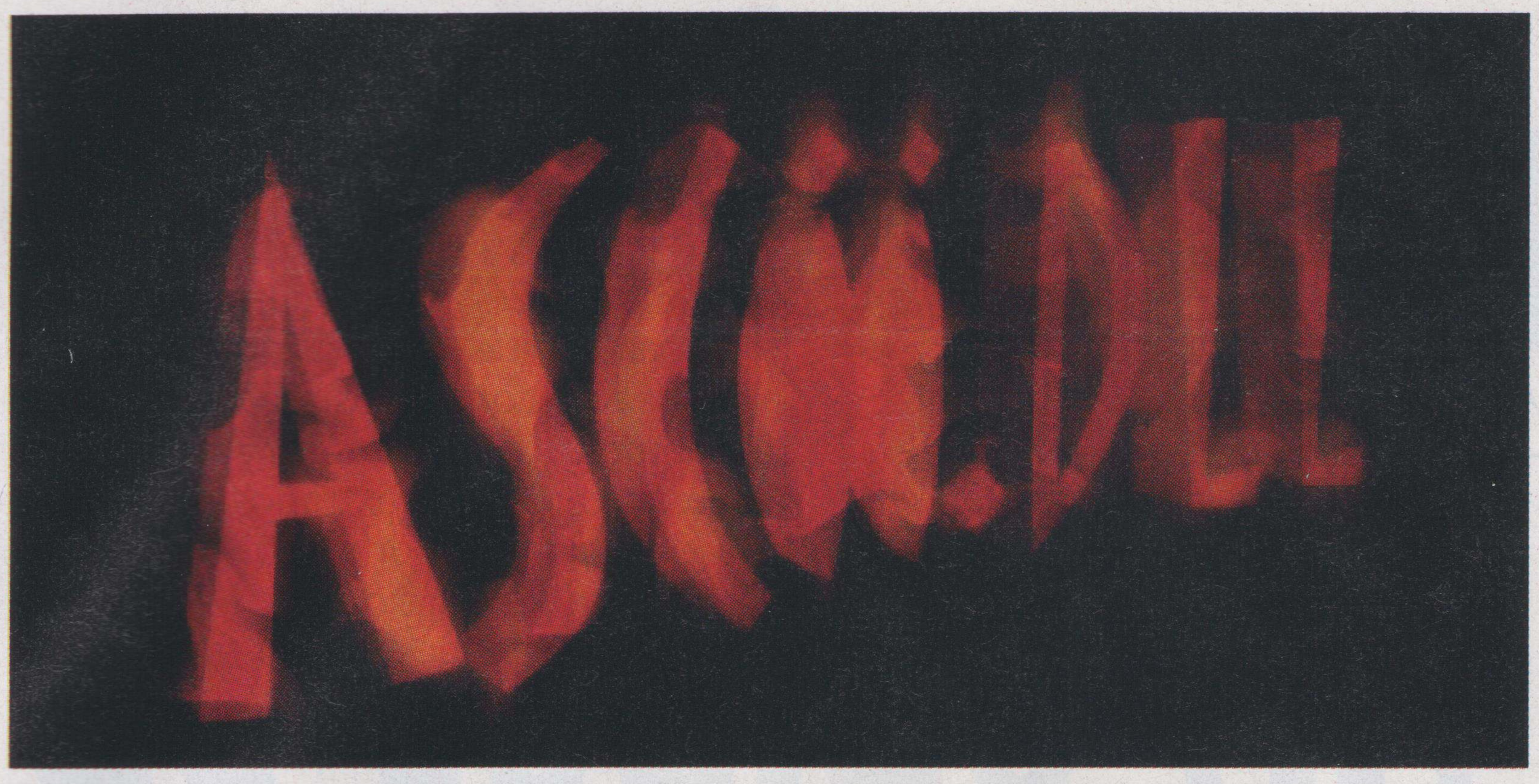
Nombre_tabla[posición] = valor;

Si queremos asignar varios valores a la vez, sólo podremos realizarlo durante la declaración, para lo cual haremos de la siguiente forma:

<tipo de dato> nombre tabla [tamaño de la tabla] = valor1, valor2,...,valor;

El valor del tamaño de la tabla puede ser mayor que el número de valores que hemos indicado posteriormente. Podemos omitir dicho valor, para de esta forma, asignar el tamaño correspondiente al número de valores que aparecen a continuación.

¿Y para qué todo esto? Pues volviendo al caso que nos preocupa, las



En la primera versión de DIV podíamos utilizar las funciones de la librería ASCII.dll para realizar operaciones que se han introducido en DIV 2.

cadenas de caracteres (o STRING), no son más que tablas en las que cada letra de nuestra cadena ocupa un lugar. Por tanto, podremos acceder individualmente a cada una de las letras como si de una tabla se tratara.

La única diferencia respecto a las tablas normales radica en el final de la cadena. Como toda tabla, una cadena tiene una capacidad límite, indicada durante la declaración. Sin embargo, podemos escribir una cadena de caracteres que ocupe menos espacio, con lo que en el espacio sobrante se almacenarán valores no deseados. Para que el compilador de DIV no tome estos valores como válidos, debemos indicar dónde acaba la secuencia de texto. Para ello utilizaremos el valor nulo, es decir, cuando nuestra cadena se acabe, añadiremos al final de ésta un valor de carácter 0.

Todo esto parece complicado, pero es completamente transparente para el programador si utilizamos las funciones que se añaden en la nueva versión de DIV para la manipulación de estas cadenas. Estas funciones, que serán descritas más adelante, manipulan sin ningún tipo de problemas las cadenas, con lo que tan sólo nos debemos preocupar de no manipularlas manualmente para no provocar fallos importantes en nuestros programas.

Veamos pues, para finalizar con esta introducción, cómo podemos definir el tamaño de nuestras cadenas de caracteres y cómo podemos asignarles un valor. Como vimos anteriormente, podemos declarar una cadena simplemente escribiendo *STRING nombre*. En este caso, el número máximo de caracteres que puede almacenar la cadena es de 256. Para variar dicha limitación, bien porque creamos excesivo el tamaño y deseemos ahorrar memoria para nuestro programa, bien porque deseamos utilizar cadenas más largas, debemos utilizar la sintaxis de declaración de tablas:

STRING nombre[tamaño];

Por tanto, STRING nombre; equivale a STRING nombre[256]; .

Nota: Los miembros de una tabla pueden ser también accedidos a través del paso de una variable que almacene un valor que indique la posición a la que queremos acceder. Esto quiere decir que el código que sigue:

```
BYTE a = 5;
Cadena[a] = "a";
```

Es equivalente a Cadena[5]= "a";

Esto nos permitirá el acceder de forma secuencial a los elementos de una tabla, por ejemplo mediante un bucle cuya variable sea el índice de nuestra tabla.

Dando valores a las cadenas

Para asignar un valor a una cadena, debemos seguir la sintaxis habitual que hemos utilizado para asignar valores a otros tipos de datos. La única diferencia radica en la expresión de la cadena. Ésta debe aparecer entrecomillada (comillas dobles) y si aparece en varias líneas, deben aparecer entrecomilladas cada una de las partes en las que está dividida la palabra y sin ningún tipo de código entre las líneas, excepto comentarios. Veamos un ejemplo:

```
STRING cadena = "DIVmanía";
```

Que es equivalente a:

```
STRING cadena = "DIV"
// cadena de caracteres "manía";
```

Realizando operaciones con cadenas

DIV 2 añade una gran cantidad de operaciones, que nos permite tratar este nuevo tipo de dato como tal, de forma que podamos realizar operaciones aritméticas de todo tipo con él. Todas estas operaciones tienen un equivalente en forma de función, si bien la forma aritmética tiene un inconveniente en lo que se refiere al tamaño de la cadena: no puede sobrepasar nunca los 1024 caracteres en el resultado. Por tanto, debemos prever siempre esta situación en todos los programas que realicemos.

Estas son las operaciones que podemos realizar:

- **Acceso como tabla:** ya descrita anteriormente.
- **Paso como parámetro:** tal vez sea la opción que nos dé más juego. Podemos enviar como parámetro a las funciones que precisen de una cadena de texto, una variable STRING. Con esta nueva capacidad podemos por ejemplo pedir el nombre de un fichero por teclado para posteriormente cargar la imagen asociada a dicho nombre mediante la función load_map. Posteriormente la analizaremos a fondo.
- **Asignación:** además del método visto anteriormente, podemos asignar el valor de dos cadenas entre sí. El código que sigue equivaldría a copiar el contenido de s2 a s1:

  S1 = s2;

  La función que equivale a esta operación es *strcpy(s1,s2)*, que recibe como parámetros las cadenas origen y destino. La cadena de origen (s2) puede ser el valor de una cadena, es decir, sería válido: *strcpy(s1,"DIVmanía");*

Los string son tablas en las que cada letra ocupa un lugar

- **Concatenación:** esta operación equivale a añadir una nueva cadena de texto al final de otra. Es decir, si tenemos dos cadenas s1 y s2 con valor *"DIV"* y *"manía"* respectivamente, la concatenación s1s2 sería "DIVmanía". El orden entre ambos operadores no se puede invertir, ya que s2s1 generaría "maníaDIV". Para realizar esta operación, utilizaremos el operador suma:

  S1 = string 1 + string 2 + ... + string n;

  Donde las cadenas pueden ser tanto un valor como una variable. La función equivalente a esta operación es *strcat(s1,s2)*, que nos permite copiar s1 al final de s2. La desventaja de esta operación es que sólo se puede concatenar un elemento al mismo tiempo.
- **Añadir caracteres:** es un caso especial del anterior. Se realiza mediante la sentencia: s1+= "a";

– **Eliminar caracteres:** se pueden suprimir del final de una cadena mediante cualquiera de las siguientes sentencias:

S1-- ; s1-=1 ; s1 = s2-1;

Una variante de esta operación la encontramos en la función *strdel(s1,prin,fin)*; donde s1 es la cadena de la que se quieren eliminar los prin caracteres y los fin caracteres.

– **Comparaciones:** nos permite determinar si una cadena es idéntica a otra o, el orden en el que aparece cada una de ellas alfabéticamente. La sintaxis es la que sigue:
S1 < s2: S1 antes en orden alfabético.
S1 <= s2: S1 antes en orden alfabético o idéntica.
S1 == s2: S1 idéntica a s2.
S1 >= s2: S1 después en orden alfabético o idéntico.
S1 > s2: S1 después en orden alfabético.
S1 <> s2: S1 no idéntico a s2.
Estas expresiones pueden utilizarse en sentencias de tipo IF. Otra opción es la de utilizar la función *strcmp(s1,s2)*; como alternativa.

Las comparaciones nos permiten determinar las cadenas idénticas

Esta función compara dos cadenas, devolviendo un número positivo si s1 va después de s2 en orden alfabético y negativo si va después. Si son idénticos, el resultado es 0.

– **Longitud:** mediante la función *strlen(s1)*; podemos determinar la longitud de una cadena. El valor devuelto no es el tamaño de la tabla que lo almacena, sino el de la palabra almacenada.

– **Relleno:** la función *strset(s1,c);* nos permite rellenar la cadena s con el carácter c repetido tantas veces como la longitud de la tabla de la cadena.

– **Posición:** las funciones *strstr(s1,s2)* y *strchr(s1,c)* devuelven la posición donde se encuentran la cadena s2 y el carácter c en la cadena s1 respectivamente.

– **Mayúsculas y minúsculas:** mediante las funciones *lower (s)* y *upper(s)* podemos convertir a minúsculas o mayúsculas respectivamente, todos los caracteres de la cadena s.

– **Cálculo de expresiones:** esta función es un tanto especial, ya que nos permite utilizar DIV como si de una calculadora se tratara. La función *calculate(s)* devuelve el resultado de realizar una operación matemática expresada en la cadena s. Dicha expresión puede ser desde un número hasta una combinación de operaciones básicas de DIV (modula, resta, suma, producto, raíces cuadradas...). Veamos un ejemplo:

```
A = Calculate ("2+3*4");
// A valdrá 14
A= Calculate("14");
// A valdra también 14
```

– **Conversión de enteros a cadenas:** mediante la función *itoa(entero)*; podemos convertir cualquier número entero en una cadena de texto. Continuando con el caso anterior, la función *s1 = itoa(A);* asignará a s1 el valor 14.

Encontrando una aplicación

Después de toda esta descripción tan abultada de datos, estamos ya en disposición de crear algo útil con todas estas funciones. Como dijimos al comienzo, podemos tomar datos desde el teclado e introducirlos en cadenas. Pero estos datos se deben leer de alguna forma. Hasta ahora, tan sólo sabemos determinar la pulsación de diversas teclas mediante la función *key()*, lo que nos supondría el realizar una función de selección tan grande para determinar la tecla pulsada, que ocuparía gran parte del tiempo de cálculo en cada frame. Es por ello que DIV posee una variable global predefinida que contiene el código ASCII (código de caracteres internacional) de la última tecla que ha sido pulsada. Esta variable se denomina *ascii*.

Este código ASCII es el que se utiliza para almacenar los diversos caracteres de una cadena y, por tanto, si tomamos dicho valor, tan sólo debemos añadirlo a una cadena como dijimos anteriormente:

```
S1 += ascii;
```

De esta forma podemos realizar una sencilla línea de comandos como la que sigue:

```
PROCESS línea_de_comandos(x,y)
PRIVATE
STRING cadena = ""; // este valor nos indica que se trata de una cadena vacía
ident;
BEGIN
Ident = write (0,x,y,0,cadena);

LOOP
  IF (key(_backspace))
    Cadena--;
  ELSE
    Cadena += ascii;
  END
  FRAME;
END
// si se sale, borra el texto
Delete_text(ident);

END
```

Hemos introducido la detección de la pulsación de la tecla de borrado para que el comportamiento de la línea de comandos sea el que tenemos acostumbrado, y no aparezcan caracteres extraños durante la ejecución. Es por ello que nos debemos siempre detectar el caso especial de la tecla de borrado.

Es importante reseñar que si bien teníamos entendido que un texto no podía variar dinámicamente como lo hacían los contadores numéricos (como vimos en los primeros capítulos con el coche de carreras), en este caso y con el uso de variables, –si disponemos de esta opción–, el código resultante es muy sencillo y se limita al cambio del contenido de las variables asignadas a los identificadores de texto.

Otra posible aplicación de estas funciones es la creación de una sencilla calculadora en línea. Para realizarla, utilizaremos las funciones *itoa* y *calculate*:

```
PROGRAM prueba;

PRIVATE
STRING cadena;
BEGIN
cadena = "";
write (0,0,0,0,cadena);
LOOP
  if (key(_backspace))
    cadena--;
  ELSE
  IF (key(_enter))
    cadena = itoa(calculate(cadena));
  ELSE
    cadena+=ascii;
  END
  END
  FRAME;
END

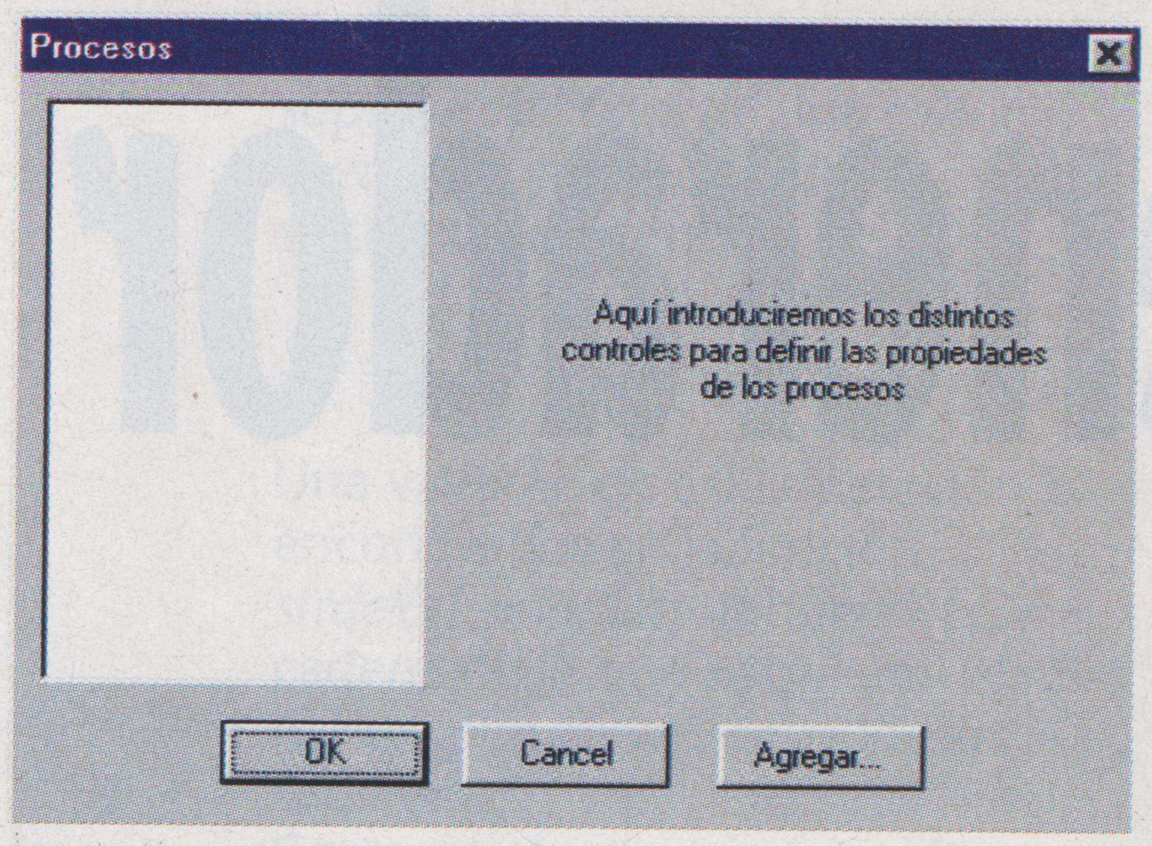
END
```

Conclusiones

Estas son algunas aplicaciones básicas de las cadenas de texto. Todas las opciones anteriormente descritas nos permiten ampliar nuestro rango de acción y realizar nuevas aplicaciones, como las tablas de récord anteriormente descritas, o el famoso juego del ahorcado, o un parser para aventuras gráficas, etc.

Para cualquier tipo de consulta, sugerencia u opinión, podéis dirigiros a la dirección de *e-mail*: *trinidad@arrakis.es.*

Pablo Trinidad



Ventana de procesos.

guardar todos los datos de una ventana en una estructura que se declare globalmente, al igual que en el caso anterior, en el archivo principal de la aplicación o en el del marco principal de la ventana. De esta forma, accedemos sin ningún problema a los datos desde cualquier punto de la aplicación, sin embargo, tenemos una tarea adicional, que es la de actualizar los datos de la ventana de diálogo cada vez que se ejecute. Para ello, utilizaremos un evento o función al que Windows llama cada vez que una ventana se inicia: *WM_INITDIALOG*. En esta función, actualizaremos las variables correspondientes a cada control, para poder mostrar el mismo aspecto que cuando salimos de dicha ventana.

Tenemos que actualizar los datos de la ventana de diálogo cada vez que se ejecute

Veamos pues el procedimiento que hemos seguido para separar dichas opciones, para lo cual utilizaremos la primera variante.

Separando las opciones

Para empezar, debemos declarar una variable en la clase *CMainFrame*, que se corresponderá con el diálogo *CDatosPrincipal*. Para ellos pulsaremos como dijimos el botón derecho sobre el nodo correspondiente a nuestra clase y seleccionaremos la opción *añadir variable miembro*. Declararemos un puntero de la siguiente forma:

```
CDatosPrincipal *datos;
```

Hemos elegido un puntero, para asignar dinámicamente memoria al diálogo, lo que es mucho más correcto que utilizar una variable estática, ya que se le reserva para ella espacio dentro del ejecutable, de forma que desde el inicio hasta el final de la aplicación se está ocupando una memoria que podría estar aprovechándose para otra cosa. Ahora, debemos asignarle memoria en la función *OnCreate*, llamada por Windows cuando se crea la aplicación. Añadimos:

```
Datos = new CDatosPrincipal;
If (Datos == NULL) return –1;
```

De esta forma nos aseguramos de que la operación se ha realizado con éxito.

Este procedimiento es el que seguiremos para cualquier diálogo que creemos en un futuro. Debemos recordar que este código no funcionará a menos que se le añada la carga del fichero de cabecera donde está declarada la clase *CDatosPrincipal* en el fichero del marco de aplicación:

```
#include "datosprincipal.h"
```

Debemos ahora separar la opción de menú en dos. Para ello, conservaremos el identificador de evento que tenía asociada la anterior opción de menú y lo reutilizaremos en la nueva opción que crearemos, donde se seguirá llamando al diálogo de datos principales. De esta forma, nos ahorramos gran cantidad de variaciones. Para ello utilizaremos el editor de recursos como especificamos en el número anterior y obtendremos dos submenús distintos:

- **Opciones:** donde están ubicadas todas las posibilidades del sistema, como los datos principales, la lista de procesos del juego...
- **Crear archivo:** donde se generará el archivo con el código definitivo de nuestro juego.

Dentro del *ClassWizard*, definiremos la función asociada a cada una de las opciones del menú. En la función asociada a la llamada del diálogo, debemos cambiar el contenido anterior (llamada estática) por una llamada tan simple como:

```
datos->DoModal();
```

Para el generador de códigos, debemos hacer algunos cambios con respecto al anterior sistema. En el número anterior, implementamos la grabación dentro de la función *OnOk()* del diálogo de datos, que se ejecutaba si se salía de la ventana de diálogo pulsando "Ok". Si copiamos el código anterior en la función asociada al evento de creación de archivo, tan sólo debemos realizar algunos pequeños cambios. Estos cambios se corresponden con la localización de las variables de datos. Estas variables ya no son locales a la clase, sino que se encuentran accesibles a través del puntero *datos*. Por tanto, el acceso debe realizarse como sigue:

```
Datos->variable;
```

Tan sólo nos falta un pequeño detalle. Debemos asegurarnos que los datos se conservan entre llamada y llamada. Para ello debemos guardarlos justo antes de salir. Esto lo haremos en la propia función OnOk(), mediante una llamada a la ya conocida función UpdateData(TRUE), que se encarga de almacenar en toda y cada una de las variables miembros, el dato almacenado en sus controles correspondientes.

Ya disponemos pues de un sistema separado.

Introduciendo el nombre del programa generado

Para poder especificar el nombre del programa que generaremos y no estar restringidos al propio uso del archivo *result.prg*, podemos realizar una operación semejante a la realizada para la selección del fichero de imágenes en el diálogo de datos básicos. Abriremos una de las ya famosas ventanas de guardar fichero, y tomaremos el nombre del fichero resultante y lo crearemos para posteriormente operar con él:

```
void
CMainFrame::OnCrearprograma()
  {
    FILE *salida;
    CFileDialog
file(false,"prg",NULL,OFN_HIDEREA-
DONLY|OFN_OVERWRITEPROMPT,
    "Programa
(*.prg)|*.prg||",NULL);
    if (file.DoModal()==IDOK)
    {
    if ( (salida =
fopen(file.GetFileName(),"w")) ==
NULL)
    {
      MessageBox("Error en la
apertura del fichero de
salida","Error");
    else
    {....
```

De esta forma sólo se grabará si pulsamos OK en la ventana de diálogo, y habremos conseguido nuestro primer objetivo.

Añadiendo nuevas capacidades

Una vez separados estos dos elementos, ya podemos plantearnos la posibilidad de ampliar nuestro entorno, añadiéndole una capaci-

dad que sin duda es bastante interesante: la creación de procesos.

Para realizar esta tarea, debemos prever el diseño que realizaremos de la misma para poder atacar frontalmente su creación. En un primer término, tendremos una ventana de diálogo a la cual se accederá desde el menú de opciones. En dicha ventana aparecerá una caja de lista, donde como su propio nombre indica, se listarán todos los procesos que queremos crear. Para crear un nuevo proceso, utilizaremos un botón donde podamos agregarlos, de forma que si se pulsa, aparecerá una nueva ventana de diálogo donde podremos establecer el nombre del proceso. Utilizaremos el segundo procedimiento de los listados al comienzo, es decir, utilizaremos una clase externa donde almacenaremos los datos de cada proceso.

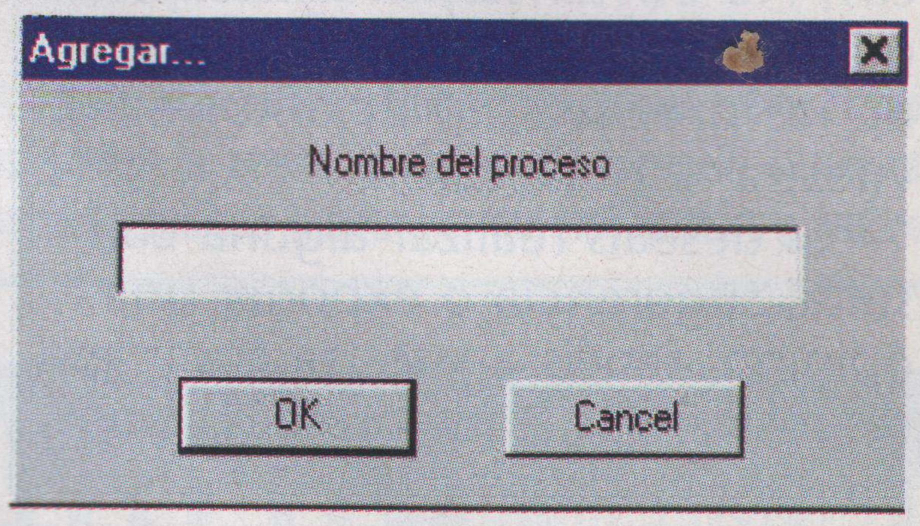
Dentro del propio diálogo, podemos especificar todas las propiedades que deseemos para cada uno de los procesos. Para ello, dispondremos de una serie de opciones que aparecerán a la derecha de dicha lista y que variarán según el proceso elegido en la propia lista. Si bien no lo realizaremos en este número, indicaremos la forma de realizarlo, ya que es muy sencillo con la estructura que vamos a definir.

Procedimiento para la creación del diálogo de procesos

Creamos una nueva ventana de diálogo, dentro de la cual introduciremos una caja de lista y un nuevo botón cuyo título será *agregar*. Mediante el *ClassWizard* crearemos la clase correspondiente a este nuevo diálogo que se llamará *CProcesosDlg*. Definimos ahora pues varios eventos que nos ayudarán a crear el comportamiento deseado tal y como podéis ver en la tabla 1.

Y debemos establecer las variables que almacenarán los valores de cada control como figura en la tabla 2.

Para guardar las propiedades de cada proceso, debemos crear



Ventana para agregar el nombre de un nuevo proceso.

Tabla 1

Control	Evento	Acción
Botón agregar	BN_CLICKED	Abre el diálogo de agregar cuando se pulsa el botón
Lista	LBN_SELCHANGE	Cambia las propiedades que aparecen en la ventana cuando se selecciona otro proceso
Botón OK	BN_CLICKED	Llama a la función UpdateData() para grabar todos los datos en sus variables correspondientes
Ventana	WM_INITDIALOG	Establece los valores de cada uno de los procesos en la ventana

Tabla 2

Control	Tipo	Variable
IDC_LISTA	CString	m_Proceso
IDC_LISTA	CListBox	m_ListaProcesos

una estructura o clase, lo que haremos rápidamente mediante la opción *Nueva clase* del menú "Insertar". Dentro de las opciones, elegiremos como tipo clase genérica y le pondremos el nombre *CProcesosDatos*. Esta clase es muy sencilla y tan sólo, contendrá una variable pública de tipo *CString* que llamaremos *m_Nombre*.

Para poder utilizar esta estructura debemos crear una pila de esta estructura, es decir, una estructura dinámica que nos permita crearlas y destruirlas a nuestro antojo. Esto es bastante más complejo y, una solución bastante rápida y funcional es crear un *array* o matriz de estas estructuras con un tamaño fijo. Dicho tamaño lo fijaremos en el momento de compilación, mediante una variable constante global o una sentencia como la que sigue:

```
#define MAX_PROCESS 10
```

Y dentro de la clase *CProcesosDlg* escribiremos:

```
class CProcesosDlg{
Public:
...
CProcesosDatos datos
[MAX_PROCESS];
Int contador;
...
};
```

La variable contador la utilizaremos para controlar en todo momento el número de estructuras de datos que se encuentran almacenadas, y evitar además añadir más procesos de los permitidos por la estructura. Es por ello que debemos iniciarla a 0 en el constructor de la clase.

Creando el diálogo de agregar

Tan sólo nos resta diseñar nuestro diálogo, donde agregaremos el nombre del proceso que deseamos crear. Para ello, utilizaremos de nuevo el editor de recursos, donde agregaremos un nuevo diálogo.

Como en todos los diálogos creados hasta el momento, en primer lugar introducimos los controles, que en nuestro caso, consistirá en una caja de edición donde introduciremos el nombre del nuevo proceso y un texto estático que nos indique el título.

Para diseñar nuestro diálogo utilizaremos el editor de recursos

En un segundo paso, creamos la clase asociada a la ventana de diálogo, a la que llamaremos *CAgregar* y le asociamos al valor del texto de la caja de edición una variable de tipo *CString* a la que llamaremos *m_Nombre*.

Por último escribiremos el código necesario. Sólo modificaremos el evento ONOK, correspondiente a la pulsación del botón de OK de nuestra ventana. De esta forma, conseguiremos que se conserve el valor introducido en la caja de edición en la variable *m_Nombre*, que posteriormente podremos leer para introducirlo en el cuadro de lista de la ventana de procesos.

Pero debemos tener en cuenta que no todos los nombre de procesos que introduzcamos son válidos. Es importante que no tengan espacios en el nombre. Los espacios pueden sustituirse por el carácter de subrayado. Por eso, el código resultante en la función *OnOK()* será:

```
void CAgregar::OnOK()
{
    UpdateData(TRUE);
    char *temp;
    temp = new
char[m_Nombre.GetLength()];
    int i = 0;
    while
(i<m_Nombre.GetLength())
    {
        if (m_Nombre[i]==' ')
            temp[i]='_';
        else
            temp[i]=m_Nombre[i];
        i++;
    };
    temp[i] = 0;
    m_Nombre = temp;
    UpdateData(FALSE);
    CDialog::OnOK();
}
```

Es importante realizar la última llamada a *UpdateData()*, ya que si no la realizamos, no se conservan los valores en la clase de forma correcta.

Ahora podremos crear nuevas opciones e integrarlas en el escritorio

Escribiendo el código

Ya sólo nos queda escribir el código del comportamiento de la clase *CProcesosDlg*:

```
void CProcesosDlg::OnOK()
{
    // Primero debemos guardar
los datos en las variables
    UpdateData(TRUE);
    CDialog::OnOK();
}

void CProcesosDlg::
OnCambioProceso()
{
```





Nuevo aspecto de la barra de menú.

```
    // aquí introduciremos los cambios en los controles de las propiedades
    // asociadas al proceso seleccionado
}

void CProcesosDlg::OnAgregar()
{
    CAgregar agregar;
    if (agregar.DoModal()==
IDOK)
    {
        if (contador < MAX_PRO-
CESS)
        {
            m_ListaProcesos.AddString
(agregar.m_Nombre);
            datos[contador].
m_Nombre = agregar.m_Nombre;
            contador++;
        }
    }
}

BOOL CProcesosDlg::
OnInitDialog()
{
    CDialog::OnInitDialog();

    for (int i = 0; i < contador;
i++)
    {
        m_ListaProcesos.AddString
(datos[i].m_Nombre);
    }

    return TRUE;
}
```

Como podemos apreciar, este código es bastante simple y tan sólo tiene una función no utilizada hasta el momento: *AddString()*. Mediante la misma, podemos añadir una cadena de texto al cuadro de lista, devolviéndonos un índice que podremos utilizar más adelante para asociarle datos mediante una función denominada *SetItemData()*. Esta asociación de datos nos permite vincular una estructura de cualquier tipo a cada una de las cadenas que aparecen en el cuadro de lista. Gracias a esto podemos asociar a cada elemento una estructura del tipo *CProcesosDatos* que contenga los valores de las propiedades de cada proceso, y se graben en la estructura global si se pulsa el botón OK y no se grabe si se pulsa Cancelar. No obstante, y por cuestiones de simplificación, esta opción la dejamos a la elección del lector. Hemos optado por agregar directamente el dato en la estructura global e incrementar el contador en consecuencia.

Modificando la creación de ficheros de salida

Para finalizar, debemos permitir la salida por fichero además de la nueva opción de creación de procesos. Para ello, debemos alterar la función correspondiente al evento de la opción de menú de *Crear archivo* y añadirle por ejemplo el siguiente código:

```
int max = Procesos->contador;
for (int i = 0;i < max; i++)
{
    fprintf(salida,"PROCESS %s()\
n",Procesos->datos[i].m_Nombre);
    fprintf(salida,"BEGIN\nLOOP\
n\tFRAME;\nEND\nEND\n\n");
}

// final del fichero
fprintf (salida,"END");
```

Con esto concluimos el desarrollo.

Conclusión

Hemos desarrollado una nueva estructura en nuestro programa que nos permite crear nuevas opciones e integrarlas en el entorno sin ningún problema y con extrema facilidad. No hemos integrado las propiedades de los procesos dentro del diálogo, sin embargo, hemos establecido una estructura que nos permite hacerlo sin ningún obstáculo. Es evidente que el sistema utilizado puede ser mejorado, sobre todo en la limitación que supone tener un número máximo de procesos fijo, mediante el uso de estructuras de almacenamiento dinámicas. No obstante y al igual que lo que ocurre con las propiedades de los procesos, lo dejamos a vuestra libre elección.

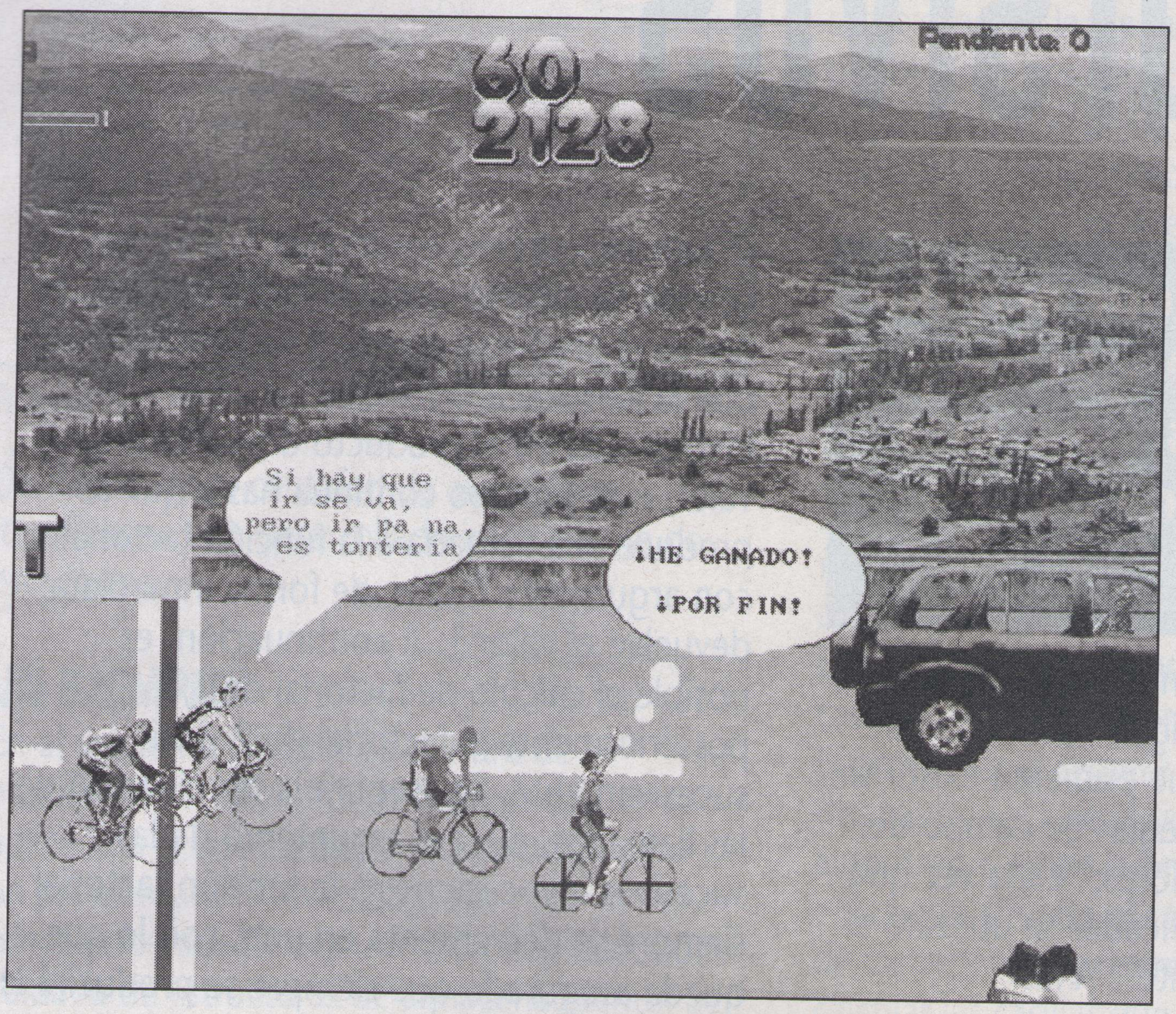
Si deseáis realizar alguna consulta, sugerencia o exponer una opinión, tenéis a vuestra completa disposición la siguiente dirección: trinidad@arrakis.es

Pablo Trinidad

DIV developer

NÚMERO 9

Sumario

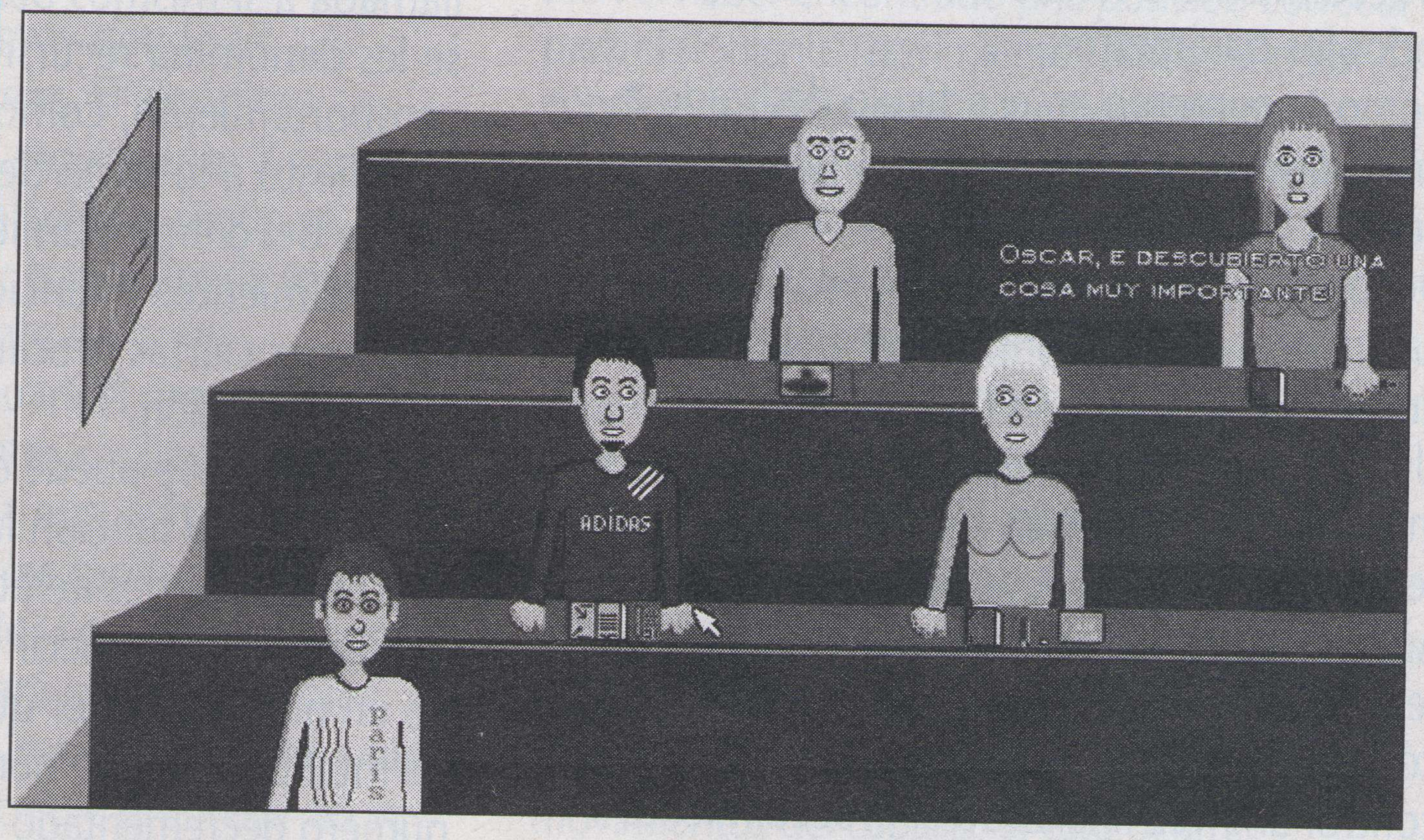


Curso de programación y concurso

Continuamos con nuestros cursos de programación habituales siguiendo los capítulos ya iniciados. Como siempre, esperamos que os sean de utilidad.
En cuanto a nuestro concurso, por supuesto sigue en pie con los mismos premios. Ya sabéis: 25.000 pesetas para el ganador y 20.000 para los otros dos juegos elegidos. Esperamos con impaciencia recibir vuestros juegos.
Aquí tenéis los ganadores de este número. Esperamos que os gusten. En este suplemento encontraréis un breve comentario del juego y un extracto de su código fuente que, no os olvidéis podéis encontrar entero en el CD-Rom si abrís los ficheros pertinentes. Seguro que la visión y el estudio de estos códigos os ayudará a mejorar algunos aspectos de vuestros propios trabajos de programación.

Vuestras sugerencias

Queremos haceros saber que todas las ideas, críticas o sugerencias para mejorar esta revista serán bienvenidas por parte de esta redacción. Si tenéis inquietud por algún tema que no tocamos demasiado, si consideráis que alguna sección de esta revista debe desaparecer o debe ser cambiada por otra, incluso si os animáis a proponernos artículos realizados por vosotros/as, que creéis pueden ser aprovechados por otros usuarios/as, no dudéis en poneros en contacto con nosotros. Todas las sugerencias serán atendidas y tenidas en cuenta. Os necesitamos para mejorar.
Para mandarnos los juegos o cualquier sugerencia sobre la publicación, podéis hacerlo por e-mail, correo normal o incluso entregarlos a mano en la dirección de la redacción:



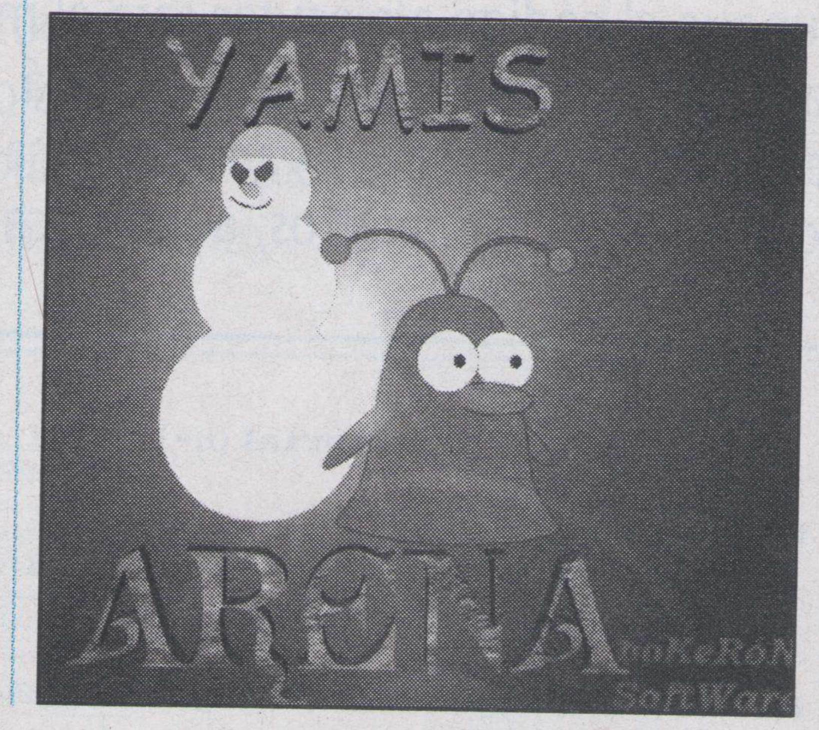
Para los juegos del concurso

No os olvidéis meter vuestros datos completos cuando mandéis juegos al concurso. Lo decimos porque hemos recibido algunos en los que falta la ciudad, el teléfono o directamente la dirección entera. Para hacer más cómoda la tarea, en el CD que acompaña la revista hay una ficha que debéis incluir como fichero de vuestro juego para evitar que haya problemas. Esta ficha se incluirá en la revista cuando hagamos otro especial de juegos DIV con todos vuestros datos.

También recordaros que es vital que introduzcáis un fichero de texto que contenga la presentación de la historia, vuestras fuentes de inspiración, las características del juego, su instalación y, sobre todo, el código fuente.

Nuestra dirección:
divmania@prensatecnica.com

Divmanía
C/ Alfonso Gómez, 42
Nave 1-1-2
28037, Madrid, España



Destacamos

En nuestro CD incluimos los juegos que han resultado ganadores en este número y, cómo no, sus respectivos códigos para que veáis todo el proceso de creación. Además, unas cuantas demos y programas que nos han enviado los lectores. Si tenéis algo que pueda resultar de utilidad, enviárnoslo y lo meteremos en el próximo CD.

Algoritmos recursivos

La recursividad es una técnica de programación que permite obtener gran cantidad de algoritmos de forma sencilla, y aunque la eficacia de éstos es, en muchos casos, baja, esta técnica sigue siendo útil como punto de partida para el diseño de algoritmos iterativos.

Cuando se hace referencia a un procedimiento o función sus parámetros se transfieren a través de la pila. Previamente a que se produzca la ejecución del código de un procedimiento o función, los parámetros se agrupan en la pila en el orden en que están declarados. Antes de devolver el control, el procedimiento o función elimina de la pila todos los parámetros.
Como se comentó en el artículo dedicado a procedimientos y funciones, los parámetros pueden transferirse por referencia o por valor. De forma simplificada, cuando se utilizan parámetros por referencia se apila un puntero que contiene la dirección actual de almacenamiento del parámetro, lo que permite modificar su contenido. Cuando un parámetro se transfiere por valor, se apila el valor actual. Además, se almacena en la pila la dirección de retorno, que no es más que el punto del código del programa al que se transferirá el control una vez finalizado el procedimiento que se invocó.

RECURSIVIDAD

El proceso que se describió con anterioridad no es incompatible con la situación en la que un procedimiento o función tiene una llamada a sí mismo dentro de su declaración. Aunque el código a ejecutar será el mismo, cada vez que se ejecute, éste lo hará teniendo en cuenta sus variables locales actuales y sus parámetros, aunque cada vez que se invoca al procedimiento se crean las anteriores variables con igual identificador, los posibles conflictos que pudieran existir se resuelven mediante las reglas de campo de validez de los identificadores: "cada vez que se referencia a un identificador se utilizará el creado más recientemente". A los procedimientos y funciones que tienen una llamada a sí mismos dentro de su declaración se les denomina recursivos. Su comportamiento, como se pondrá de manifiesto a lo largo de este artículo, no difiere del comportamiento habitual de los procedimientos y funciones, aunque habrá que tener ciertas precauciones a la hora de su implementación.
En la figura 1 se puede observar una definición recursiva de la función factorial junto con sus implementaciones iterativa y recursiva. La definición viene a decir lo siguiente: el factorial de un número mayor que cero es el resultado de multiplicar dicho número por el factorial del número decrementado en 1, mientras que el factorial del número 0 es 1. La recursividad de la definición es evidente, pues para definir la función factorial se recurre a la misma función. Es importante señalar que el caso del número 0 hace que dicho algoritmo sea capaz de terminar, pues mediante las definiciones recursivas es posible construir algoritmos que no finalicen, por tanto, siempre que se implemente uno de estos algoritmos será necesario buscar una condición que a lo largo de las llamadas recursivas se verifique y que sirva para detener las llamadas al procedimiento o función.
En la figura 2 puede observarse la sucesión de llamadas que se producen a la hora de llamar a la función factorial con el parámetro 4. Como era previsible, la operación producto que aparece en el cuerpo de la función queda suspendida momentáneamente, hasta que se resuelve la llamada a la función factorial con el argumento decrementado en 1, a su vez, cuando se intenta ejecutar dicha función de nuevo el producto quedará suspendido hasta que la llamada a la función factorial involucrada en el producto devuelva un valor; este proceso continúa hasta que se produce una llamada a la función factorial con argumento 0 que, de forma inmediata, devuelve el valor 1. A continuación, es posible el cálculo de factorial de 1, posteriormente el factorial de 2 y así sucesivamente. Se asegura la terminación de las llamadas, pues el argumento de las sucesivas invocaciones a la función factorial siempre se decrementa en uno, con lo que queda asegurado que siempre se producirá la llamada a *factorial(0)*.
La versión iterativa para la función factorial no hace más que almacenar en la variable *acum*, los resultados intermedios que se producían en la versión recursiva. Como se mostrará en el artículo, los algoritmos recursivos se pueden transformar en algoritmos iterativos.
En la figura 3 se muestra la definición y los algoritmos iterativo y recursivo para la función de Fibonacci; en este ejemplo se pone de manifiesto que la versión iterativa es, con mucho, más eficiente que la versión recursiva, pues en ésta cada llamada a la función provoca dos nuevas llamadas y así sucesivamente, por lo que el número de invocaciones crece exponencialmente. Este ejemplo no debe hacer pensar que la recursión es una mera curiosidad, pues es posible comenzar el diseño de un algoritmo utilizando una solución recursiva. Si tras su estudio se comprueba que no es factible su uso debido a las restricciones de tiempo o espacio impuestas a éste, es posible implementar una pila para recoger los resultados intermedios y proceder a implementar el algoritmo de forma iterativa.

```
                 ┌ n*factorial(n-1) , si n > 0
factorial(n) =   │
                 └ 1                , si n= 0

function factorial (n : integer) : integer ;
 begin
  if n= 0
    then
     factorial := 1
    else
```

FIGURA 1.

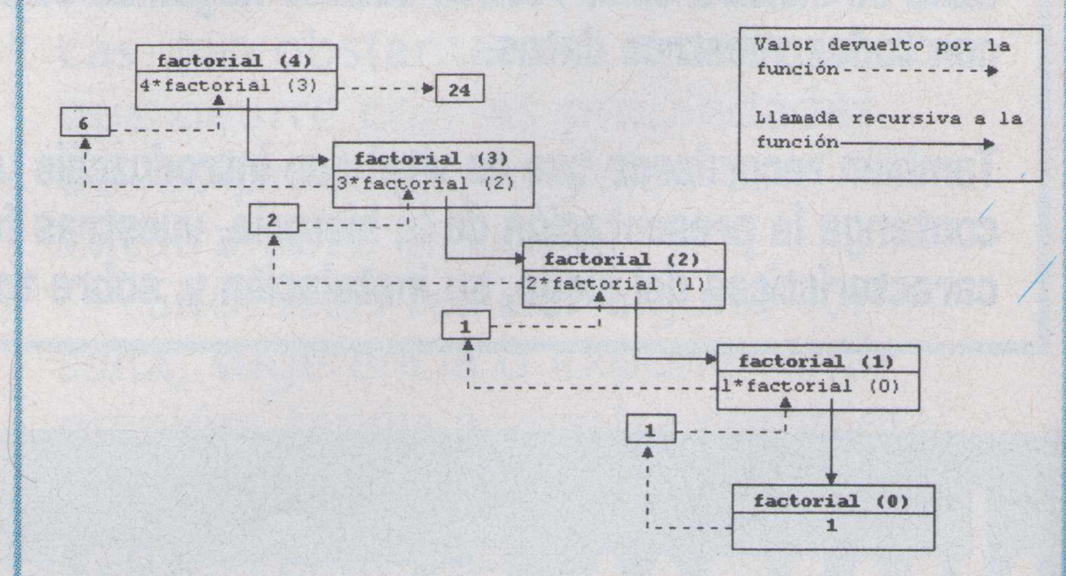


FIGURA 2.

```
function Fibonacci (n : integer) : integer ;
 begin
  if n = 0
    then
     fibonacci := 0
    else
     if n= 1
       then
        fibonacci := 1
       else
        fibonacci := fibonacci(n-1) + fibonacci (n-2) ;
 end ;

function Fibonacci (n : integer) : integer ;
 var
  resul, anterior, i : integer ;
 begin
  resul := 1 ;
  anterior := 0 ;
  i := 1 ;
  while i < n
       do
        begin
          resultado := resultado + anterior ;
          anterior := resultado - anterior ;
          i := i + 1 ;
        end ;
  fibonacci := resultado ;
 end ;
```

```
Definición de la función de
         Fibonacci

fibonacci(0)= 0

fibonacci(1)= 1

fibonacci(n+1)= fibonacci(n) +
                fibonacci(n-1)
```

FIGURA 3.

De hecho, gran cantidad de algoritmos iterativos se han desarrollado inicialmente de esta forma.

UN EJEMPLO

En el siguiente ejemplo se pondrá de manifiesto la utilidad de la recursividad aplicada a la búsqueda de una solución para un problema que no se basa en procedimientos de cálculo ni en una selección cuidada de estructuras de almacenamiento. Se trata de recorrer un tablero de *n* casillas de ancho por *n* de largo utilizando los movimientos de un caballo de ajedrez situado inicialmente en una casilla dada.

En principio, no es posible encontrar una fórmula de cálculo que haga corresponder a cada casilla un número que represente el orden en que tal casilla ha sido seleccionada durante el camino recorrido por el caballo. A falta de dicha fórmula se va a intentar situar en una de las casillas a la que es posible moverse, realizando este proceso repetidas veces. Es evidente que la mayoría de las veces no será posible continuar en un determinado punto, quedando casillas del tablero por las que el caballo aún no ha pasado. Una solución consiste en guardar de alguna forma las casillas que el caballo podría haber utilizado para sus anteriores movimientos y no utilizó, intentando seleccionar una de dichas casillas, realizando nuevos movimientos hacia delante y así poder encontrar la solución. Este tipo de algoritmo se denomina "ensayo y error" y como se podrá comprobar la recursividad será una ayuda muy importante.

Se comenzará la solución del problema seleccionando las estructuras de datos a utilizar. Como es obvio, el tablero estará representado por un array bidimensional de *n* filas por *n* columnas cuyo tipo base será un entero positivo. El valor 0 de una componente representará que la casilla correspondiente no ha sido utilizada; cada vez que una casilla sea visitada por el caballo se le dará el valor del número de movimiento correspondiente (hay n^2-1 movimientos a realizar en total), de tal forma que al finalizar el algoritmo puedan seguirse el orden de los movimientos.

La siguiente cuestión a resolver consiste en establecer la representación para los movimientos del caballo y establecer un orden sobre ellos. En la figura 4 se muestran los posibles movimientos del caballo numerados según el sentido de las agujas del reloj. Para calcular los posibles movimientos del caballo se utilizará un array de 8 elementos de tipo registro con dos componentes enteros que indican los incrementos y decrementos a aplicar sobre la posición actual del caballo. Hay que indicar que estos movimientos son todos los posibles, no los factibles, pues si el caballo está situado suficientemente cerca de los límites del tablero no será posible realizar alguno de los movimientos dentro de dichos límites; de igual forma no serán posibles los movimientos a las casillas que se ocuparon previamente.

El proceso que seguirá el algoritmo que se está diseñando será de la siguiente forma: el algoritmo conoce la primera casilla donde se encuentra situado el caballo y a través del array de incrementos y decrementos es capaz de calcular las casillas que le es posible ocupar en el siguiente movimiento, además, éstas siguen el orden del índice del array donde se almacenan los incrementos. El algoritmo intentará probar para el movimiento 2 con la primera de las casillas calculadas según se describió anteriormente; tras ocuparla (dando el valor 2 a la componente del array correspondiente) continuará el proceso de ocupación de las siguientes casillas. Este proceso continuará hasta que se hayan ocupado la totalidad de las casillas (el número de movimientos realizados será de n^2-1) con lo que el algoritmo terminará con éxito, o bien, al intentar ocupar la casilla correspondiente a un movimiento *k* no le es posible progresar en su recorrido (ninguno de los movimientos permitido es factible), el algoritmo tendrá que deshacer la ocupación de la casilla correspondiente al movimiento *k-1* y volver a intentar ocupar una casilla para el movimiento *k*. Este proceso puede involucrar a la casilla ocupada en el movimiento *k-2*, si ninguno de los movimientos factibles realizados en el movimiento *k-1* tienen como resultado el poder ocupar una casilla en el movimiento *k*; este proceso puede provocar un efecto dominó hacia movimientos anteriores. El algoritmo terminará cuando se han analizado todas las posibles combinaciones (no se ha encontrado ninguna solución) o bien cuando se encuentre la primera solución (se cubrieron todas las casillas del tablero).

Para implementar el algoritmo se ha decidido que el array que representa el tablero y el array de incrementos sean variables globales, pues es inadmisible, dado su tamaño, que cada vez que se llame al procedimiento recursivo éstos pasen como parámetros. Sí que formarán parte de los parámetros del procedimiento el número de movimiento que se va a realizar y la posición desde la que se debe realizar dicho movimiento pues, a partir de ésta, se calculan los posibles movimientos hacia la siguiente casilla. Falta decidir cómo es posible comunicar al procedimiento que provoca la llamada recursiva si se ha realizado con éxito el movimiento que se solicita a través de ésta. Bastará con pasar por referencia un parámetro de tipo booleano.

La variable *i_incremento* será utilizada para recorrer el array que contiene los incrementos, para calcular los posibles movimientos desde la coordenada que le es dada a modo de parámetro. La variable *pos_prueba* contiene la coordenada desde la que realizará el siguiente movimiento, y la variable *aux_ok* permitirá conocer el resultado del siguiente movimiento, correspondiente a la llamada recursiva. El cuerpo del procedimiento no es más que un bucle que tiene como condición de salida el haber intentado los ocho posibles movimientos desde la posición que le es pasada como parámetro o bien el haber encontrado una posición correcta para el siguiente movimiento (ambas son las condiciones de parada del procedimiento recursivo), o bien el haber realizado el último movimiento.

En el cuerpo correspondiente a la instrucción iterativa se puede observar cómo se procede a ocupar la primera casilla que se encuentra disponible, anotando dicho movimiento en el tablero, y realizando una llamada recursiva para que se realice el siguiente movimiento a partir del movimiento realizado. Esta llamada recursiva puede devolver el valor *false* en el parámetro *ok_aux* indicando que no pueden progresar los movimientos del caballo. Si esto ocurre, entonces el procedimiento intenta realizar un nuevo movimiento (esto es posible puesto que en la variable *i_incremento* se almacena la posición del último incremento utilizado para el cálculo del movimiento) y seguir realizando la llamada recursiva. Pero puede ocurrir que no queden más movimientos por realizar; en este caso, el procedimiento devolverá en el parámetro *ok* el valor *false*, de tal forma que desde el punto en el que fue llamado se intente un nuevo movimiento.

Posición	Incrementos
1	(2, -1)
2	(2, 1)
3	(1, 2)
4	(-1, 2)
5	(-2, 1)
6	(-2, -1)
7	(-1, -2)
8	(1, -2)

Valores para el array de movimientos

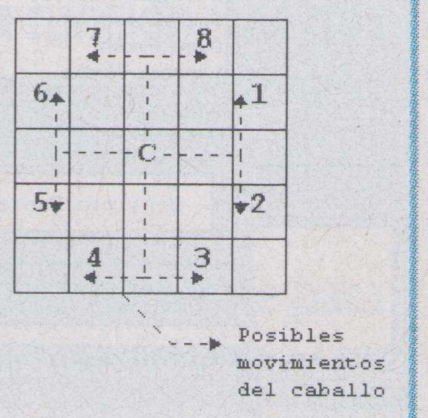
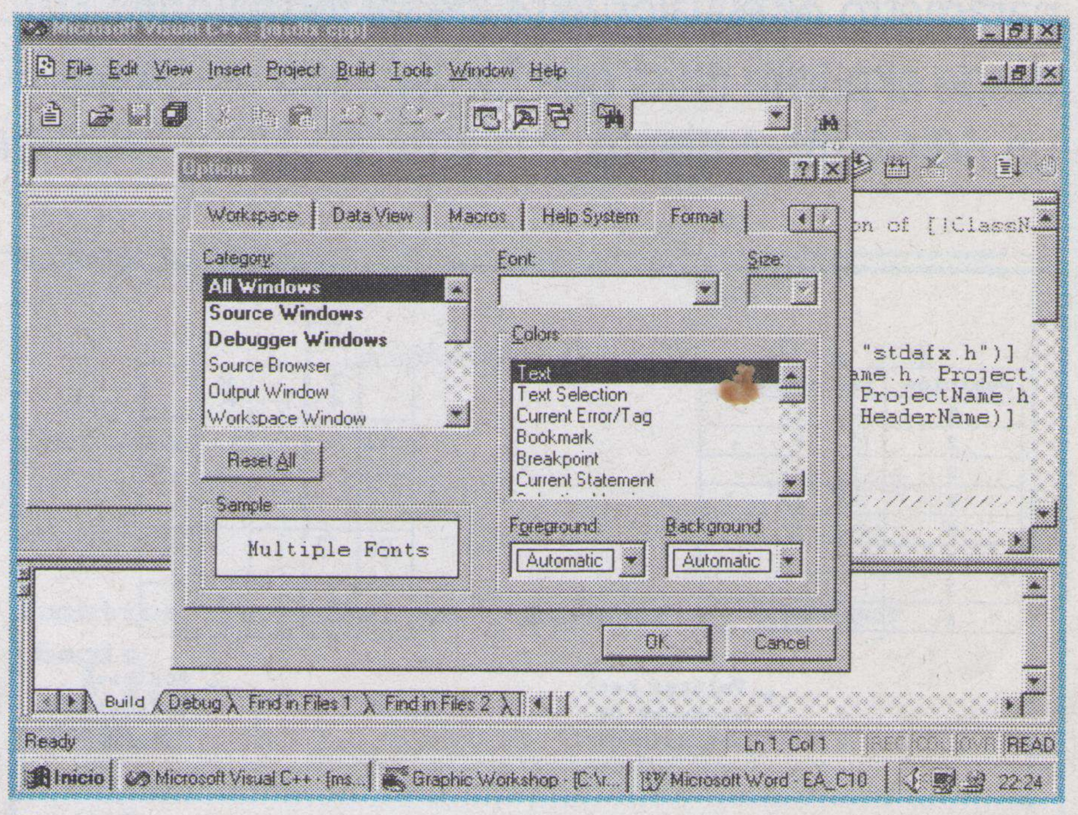


FIGURA 4.

Cómo usar punteros en C

En este artículo se presenta el primer capítulo de un pequeño cursillo de C que se va a dedicar al tema de los punteros, un tema muy interesante e importante que sirve para conocer y poder realizar aplicaciones potentes y rápidas.

La mayoría de la gente considera los punteros como uno de los temas más difíciles del lenguaje C. Existen diferentes razones para que esto ocurra. Primero, el concepto de puntero lleva emparejado otro, el de dirección, que puede ser nuevo para los programadores, ya que no es de uso común en lenguajes como BASIC o Pascal. Segundo, los símbolos utilizados en la notación de punteros no son tan claros como debieran; por ejemplo, se utiliza el mismo símbolo para dos propósitos diferentes aunque relacionados. Conceptualmente, sin embargo, los punteros no son realmente tan complicados y con un poco de práctica los símbolos comenzarán a tener sentido. En otras palabras, los punteros pueden ser difíciles, pero no demasiado. Nuestro objetivo en este minicurso dedicado a los punteros es desmitificarlos, explicando tan claramente como sea posible qué hacen y cómo trabajan.
Para cumplir este cometido, empezaremos lentamente, en un intento de asegurar la comprensión de los conceptos básicos antes de profundizar en el estudio de los punteros en situaciones más avanzadas.
Antes de ver ejemplos de programas que utilicen punteros, vamos a realizar un examen global de este concepto y de los motivos de su empleo.



En un compilador actual de Windows, todo es configurable.

¿QUÉ ES UN PUNTERO?

Un puntero proporciona una vía de acceso a una variable (o a tipos de datos más complejos, como por ejemplo un *array*) sin referirse directamente a la variable. Para ello se emplea la dirección de variable. En efecto, la dirección actúa como un intermediario entre la variable y el programa que quiere acceder a ella. En el negocio del espionaje existe un mecanismo análogo: un agente puede dejar su información en un lugar especial (un buzón de correos o un agujero en un árbol) y no tener contacto directo con los otros miembros de la red. Así, si es capturado, la información que el agente pueda revelar bajo tortura es muy pequeña. Podemos decir que el agente tiene sólo acceso indirecto a aquellos a los que proporciona información.
De una forma similar, una instrucción del programa puede referirse indirectamente a una variable, utilizando la dirección de ésta como una especie de buzón de correos para pasar la información.

¿POR QUÉ SE UTILIZAN LOS PUNTEROS?

Los punteros se utilizan en situaciones en las que pasar valores reales es difícil o indeseable (es raro el caso en el que un programa enemigo fuerza a una función a revelar los nombres de las variables utilizadas). Algunas razones para utilizar punteros son:

1. Que una función devuelva más de un valor.
2. Pasar, de una forma más adecuada, *arrays* y cadenas de una función a otra.
3. Manipular *arrays* más fácilmente moviendo los punteros que les apuntan (o que apuntan a algunos de sus elementos), en lugar de mover los propios *arrays*.
4. Crear estructuras complejas de datos, tales como listas indexadas y árboles binarios, donde una estructura de datos puede contener referencias a otras estructuras.
5. Comunicar información sobre la memoria, como con la función *malloc( )*, que informa sobre la localización de memoria libre utilizando un puntero.

De momento vamos a explorar algunos de estos usos de los punteros.Otra razón esgrimida con frecuencia para justificar el uso de los punteros es que la notación con punteros se compila en código más rápido y eficiente que, por ejemplo, la notación de *arrays*. No está claro que éste sea un factor importante para los modernos compiladores; probablemente muchos programadores se acaben *enamorando* de la notación de punteros y abrazarán cualquier excusa para poder usarlos.

Devolución de datos desde las funciones

Comenzaremos nuestro examen sobre punteros descifrando cómo las funciones devuelven valores múltiples a los programas que las llamaron. Ya hemos visto que es posible pasar valores a una función y que ésta devuelva un valor simple al programa, pero ¿qué ocurre cuando quiere pasar más de un valor desde una función al programa? Como las funciones no cuentan con los mecanismos necesarios para realizar esta tarea, debemos confiar en los punteros.

Paso de valores a una función

Antes de mostrar cómo trabajan los punteros, revisemos qué ocurre cuando pasamos valores *a* a una función. A continuación se muestra un ejemplo muy simple que pasa dos valores, los enteros 4 y 7, a una función llamada *gets2( )*:

```
/* valores.c */
/* funciones de prueba con dos valores aceptados
*/
void gets2(int, int);
main( )
{
  int x=4, y= 7;
  gets(x, y);
}

/* gets2( ) */
/* imprime los valores de dos argumentos */
void gets2 (int xx, int yy)
{
  printf("El primero es %d, el segundo es %d", xx,
yy);
}
```

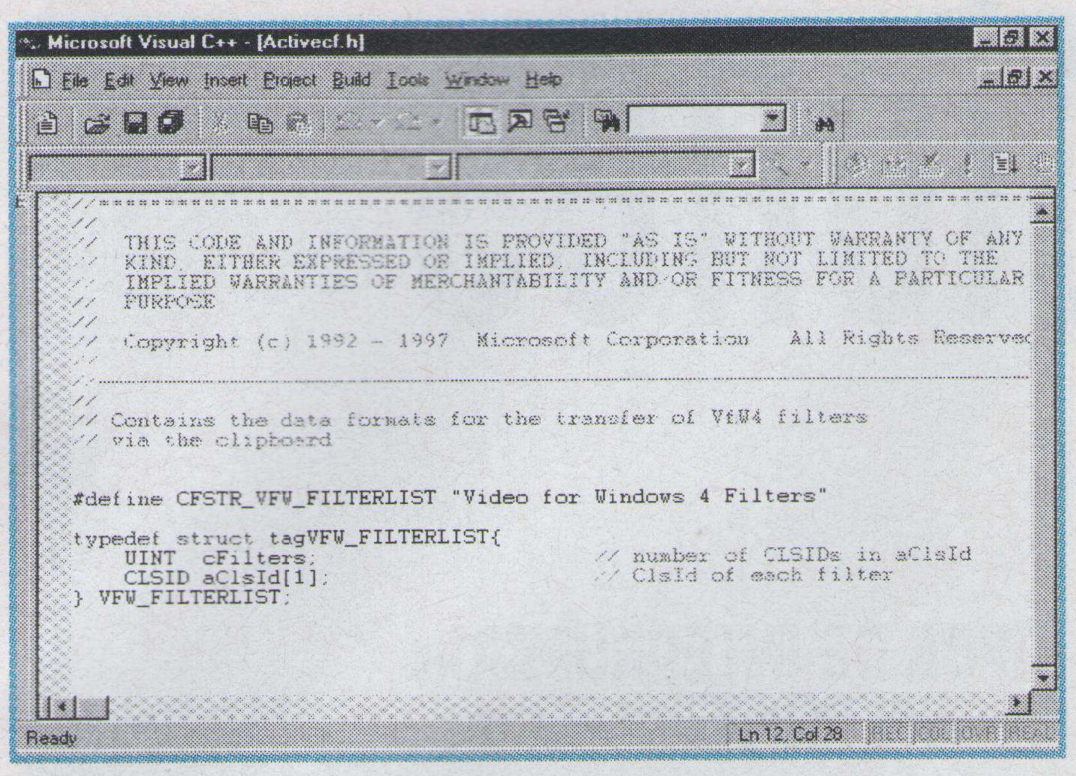


FIGURA 1.

No es una función muy útil: simplemente imprime los dos valores que se le pasan. Sin embargo, demuestra un punto importante: la función recibe los dos valores pasados por el programa y los almacena, o mejor, almacena duplicados de ellos en su espacio privado de memoria. De hecho, puede dar nombres diferentes a esos valores conocidos sólo por la función: en este caso, *xx* e *yy*, en lugar de *x* e *y*. La función puede manejar las nuevas variables, *xx* e *yy*, sin afectar a las originales, *x* e *y*, del programa.

Paso de direcciones a una función
Veamos ahora el caso contrario: pasar dos valores de la función al programa que la llamó. ¿Cómo podemos hacer esto? Se utiliza un proceso de dos escalones. Primero, el programa, en lugar de pasar valores a la función, pasa sus direcciones.
Éstas son las direcciones de las variables del programa donde se quieren almacenar los valores retornados. Veamos el programa:

```
/* direccio.c */
/* función de prueba que devuelve dos valores */
void dev2(int *, int *);

main( )
{
 int x, y;
   dev2(&x, &y);
  printf("El primero es %d, el segundo %d", x, y);
}

/* dev2( ) */
/* imprime dos números */
void dev2(int *px, int *py)
{
  *px = 3;
  *py = 5;
}
```

El programa tampoco realiza nada demasiado útil. El programa *main( )* llama a la función *dev2( )*, que devuelve dos valores, 3 y 5, al programa. Posteriormente, el programa imprime estos valores. A pesar de su aparente inutilidad, el listado está lleno de nuevas ideas. Veámoslas paso a paso.
Primero, observamos que el programa no da ningún valor a las variables *x* e *y*. Pero cuando se ejecuta el programa, estas variables tienen un valor, como podemos ver en la ejecución, ya que es el programa el que las imprime. Podemos inferir que la función *dev2( )* suministra, de alguna forma, estos valores al programa.
El programa le dijo a *dev2( )* dónde poner los valores al pasar las direcciones de las variables. La forma de conseguir esto es mediante el empleo del operador de dirección &. La expresión

Dev2(&x, &y);

pasa las direcciones de *x* e *y* a la función, que las almacena en su espacio privado de memoria. La función da nombre a estas direcciones: *px* y *py*. De esta forma, la función puede referirse a ellas, como si fueran otro tipo de variables (naturalmente podríamos haber utilizado otros nombres, por ejemplo *punt_a_x* y *punt_a_y*, que son más descriptivos pero más largos).

Declaración de variables de puntero
Como sucede con otros tipos de variables, se deben declarar los lugares reservados para estas direcciones, *px* y *py*, ya que el compilador debe saber sus nombres y cuánto espacio de memoria tiene que asignarles. Como estamos almacenando direcciones, o constantes de punteros, se supone que existe otro tipo de dato, diferente a los vistos hasta ahora. Algo como lo siguiente:
Ptr px, py

Siendo *ptr* el tipo de dato para los punteros. En general, todas las direcciones tienen el mismo tamaño y queremos reservar la memoria suficiente para almacenar una dirección. Normalmente, dos bytes son suficientes para almacenar una dirección (cuando se utilizan modelos de memoria distintos al modelo *pequeño*, esto puede dejar de cumplirse; sin embargo, todos los ejemplos contenidos en este cursillo utilizan el modelo pequeño).
Al declarar una variable de puntero reservamos dos bytes de memoria, pero existe una complejidad añadida. Por motivos que explicaremos más adelante, el compilador necesita conocer no sólo que hemos declarado un puntero, sino a qué tipo de dato está apuntando el puntero. En otras palabras, cada vez que reservamos espacio para almacenar la dirección de una variable, necesitamos decirle al compilador el tipo de dato de esta variable. Especulemos otra vez con la posible apariencia que tendría la declaración de un puntero:

Int _ptr px, py;

Siendo *int _ptr* el tipo de dato para los punteros que apuntan a variables enteras. Nos vamos acercando. Sin embargo, C es un lenguaje conciso; por tanto, en lugar de utilizar la palabra *ptr*, emplea el asterisco. El asterisco actúa como diferenciador de las palabras que representan tipos de datos simples (por ejemplo, *int* y *float*) y se utiliza inmediatamente antes de cada *variable*, en lugar de utilizarse una sola vez al principio de la declaración. De esta forma la declaración para dos punteros a enteros es:

*Int *px, *py;*

La declaración reserva dos bytes en los que se almacena la dirección de una variable entera y denomina a este espacio de almacenamiento con el nombre *px*. También reserva otros dos bytes en los que almacena la dirección de otra variable entera y le da el nombre de *py*. Los asteriscos dicen al compilador que estas variables contienen direcciones (no valores) y la palabra *int* indica que estas direcciones apuntan a variables enteras. Observamos que la declaración no dice nada sobre lo que se colocará en esas variables.

El estilo conciso de este formato de declaración es uno de los motivos de confusión con los punteros. Por tanto, reiteramos lo que ocurre: para cada variable (*px* y *py* en este caso) la declaración hace que el compilador reserve un espacio en memoria de dos bytes en los que se pueden colocar sus direcciones.
Además, el compilador es consciente del tipo de variable a la que se refiere la dirección, en este caso enteros.

Pasar valores a las variables de punteros
El programa llama a la función mediante la instrucción:

dev2(&x, &y);

las direcciones proporcionadas por el programa, *&x* y *&y*, se colocan en los espacios indicados en la declaración de la función. De esta forma, se pasa el control y las dos direcciones (que pueden ser 1310 y 1312) a la función. Las dos direcciones se colocan en el espacio reservado para *px* y *py*.

Examinemos este proceso cuidadosamente para asegurarnos que estamos en el buen camino. Podemos decir que *px* y *py* son variables de punteros y que las direcciones 1310 y 1312 son constantes de punteros.

CURSO DE ENSAMLADOR

Usando las interrupciones

Las interrupciones son parte constituyente del propio diseño del microprocesador y son las que hacen posible que funcione el teclado, el reloj interno, la BIOS y el propio sistema operativo. En este capítulo se va a detallar cómo poder acceder a esta completa colección de rutinas.

Las interrupciones, desde el punto de vista práctico del programador, son las rutinas que están instaladas en el sistema y que están a nuestra disposición como si de una gran librería de funciones residentes en memoria se tratara y las cuales podemos utilizar en cualquier momento desde nuestras aplicaciones.
Estas funciones se clasifican en grupos según su utilidad, teniendo, así, que disponemos de rutinas para escribir en pantalla, para leer y escribir en disco, para gestionar la memoria y otras funciones de mucha utilidad y que, en caso de tener que realizar nosotros mismos directamente mediante código *a mano*, haría que la programación en ensamblador fuese un verdadero infierno, ya que habría que conocer tan a fondo el hardware de la máquina como los propios creadores del mismo. Así, por ejemplo, simplemente para realizar la operación de crear un fichero en el disco, en caso de no disponer de rutinas destinadas a ello, habría que realizar un gigantesco programa que tendría que acceder directamente a nivel binario con la controladora de disco, para lo cual se necesitarían conocimientos completos del funcionamiento interno de la misma.

CLASIFICACION DE LAS INTERRUPCIONES

En total existen 256 interrupciones, que se numeran desde siempre por costumbre en hexadecimal (así lo veremos en todos los libros), por lo que hay interrupciones desde la 00h hasta la Ffh.
Estas 256 interrupciones están divididas en cuatro grandes grupos, que son: las interrupciones BIOS, las interrupciones DOS, las de usuario (libres) y las del BASIC (que llevaban los primeros ordenadores IBM PC en la ROM y el cual se ejecutaba automáticamente cuando no se encontraba un disco de arranque).
Cada uno de estos grupos mencionados se divide, a su vez, en subclases según su utilidad. En la **tabla 1** puede verse un mapa de todas las interrupciones.

TIPOS DE INTERRUPCION SEGUN ORIGEN

Independientemente de los cuatro grupos a los que pertenecen todas las interrupciones (BIOS, DOS, Usuario y BASIC), todas ellas se refieren, a su vez, a tres clases dependiendo de si son ejecutadas cuando lo pide la propia CPU, si lo pide un dispositivo físico externo o si sólo pueden ejecutarse explícitamente desde código de programa. Según lo dicho, se llamarán excepciones, interrupciones hardware o interrupciones software.
Las excepciones son aquellas interrupciones que se ejecutan cuando ocurren situaciones anómalas de funcionamiento de la CPU.

TABLA 1: MAPA DE INTERRUPCIONES DEL SISTEMA

Interrupciones BIOS:
00h- 07h: Interrupciones internas del microprocesador.
- 00h: División por cero.
- 01h: Ejecución paso a paso.
- 02h: No enmascarable.
- 03h: Breakpoint.
- 04h: Overflow.
- 05h: Imprimir pantalla.
- 06h-07h: No usadas.

08h- 0Fh: Interrupciones del controlador 8259.
- 08h: Temporizador.
- 09h: Teclado
- 0Ah-0Dh: No usadas.
- 0Eh: Diskette.
- 0Fh: No usada.

10h- 1Ah: Puntos de entradas de la BIOS.
- 10h: Entrada/Salida Pantalla.
- 11h: Chequeo del equipo físico.
- 12h: Tamaño de la memoria.
- 13h: Diskette.
- 14h: Entrada/Salida Puerto serie.
- 15h: Entrada/Salida cassette.
- 16h: Entrada/Salida Teclado.
- 17h: Entrada/Salida Impresora.
- 18h: Llamada al BASIC de cassete.
- 19h: Reinicializar el sistema.
- 1Ah:Contador del temporizador.

1Bh- 1Ch: Rutinas del usuario.
- 1Bh: Rutinas del teclado.
- 1Ch: 'Tic' del temporizador.
- 1Dh: Inicialización de vídeo.
- 1Eh: Parámetros de diskette.
- 1Fh: No se usa.

1Dh- 1Fh: Parámetros de la BIOS.

Interrupciones DOS:
20h: Terminación del programa.
21h: Petición de una función.
Servicios disponibles:
- AH=0h a 12h: Entrada/Salida Carácter.
- AH=13h a 24h: Gestión de ficheros.
- AH=25h a 26h: Funciones varias.
- AH=27h a 29h: Gestión de ficheros.
- AH=2Ah a 2Eh: Funciones varias.
- AH=2Fh a 38h: Sólo a partir del DOS 2.0.
- AH=39h a 3Bh: Gestión de directorios.
- AH=3Ch a 46h: Gestión de ficheros

22h: Dirección de finalización del programa.
23h: Dirección de salida con CTRL+BREAK
24h: Rutina gestora de errores críticos.
25h: Lectura absoluta de disco.
26h: Grabación de disco absoluta.
27h: Terminar programa dejándolo residente.
28h- 3Fh: Reservadas, no usadas.

Interrupciones de Usuario:
40h- 7Fh: Libres para uso del programador.

Interrupciones B.A.S.I.C.
80h- Ffh: Funciones del BASIC
(si está presente)

TABLA 2: MAPA DE INTERRUPCIONES DEL SISTEMA

00h: Se ha producido una división por cero.
01h: Paso a paso. Cuando el modo depuración está activado se ejecuta tras cada instrucción.
03h: Punto de ruptura encontrado. Usado en modo de depuración.
06h: Se ha producido un Código de operación inválido.
07h: Coprocesador matemático no presente.

Ejemplo de ello es cuando se hace una división por cero, cuando la CPU encuentra un código de operación no válido, etc. En la **Tabla 2** hay un mapa de las excepciones que existen.
Las interrupciones hardware son aquellas que están reservadas para dispositivos físicos del sistema, como el teclado, unidades de disco, el reloj interno, etc, y que se ejecutan automáticamente cada vez que el dispositivo físico asociado (correspondiente) envía una señal de petición de interrupción *hard* (como ocurre, por ejemplo, cuando pulsamos una tecla). En la **Tabla 3** podemos ver una lista de las interrupciones hardware.
Por otro lado, tenemos que las interrupciones software son las que están disponibles para el programador y/o sistema operativo, y que sólo se ejecutan si las llamamos explícitamente mediante una instrucción que hay para tal fin y que hace las funciones de la instrucción *CALL* que, como deberíamos recordar de anteriores números, se usaba para llamar a procedimientos normales dentro de nuestro propio programa.

TABLA 3: MAPA DE INTERRUPCIONES HARDWARE

08h: Interrupción del temporizador.
09h: Teclado, se ejecuta con pulsación de tecla.
0Ah: Canal 1/0.
0Bh: Señal desde el COM2.
0Ch: Señal desde el COM1.
0Dh: señal desde LPT2, impresora.
0Eh: Señal enviada por controladora de diskette.
0Fh: Señal enviada por LPT1, impresora.
70h: Reloj de tiempo real.
71h: Desvío de la IRQ2.
72h: Reservada para el sistema.
73h: Reservada para el sistema.
74h: Reservada para el sistema.
75h: Señal procedente del Coprocesador (FPU).
76h: Señal de la Controladora de disco duro.
77h: Interrupción reservada para el sistema..

EL PRIMER GRUPO, INTERRUPCIONES BIOS

Las interrupciones BIOS son un total de 32 y comprenden desde la primera, la 00h hexadecimal, hasta la 1Fh, y se dividen en cinco subclases: las interrupciones internas del microprocesador (00h a 07h), las del controlador de interrupciones 8259 (08h a 0Fh), los puntos de entrada de la BIOS (10h hasta 1Ah), las rutinas del usuario (1Bh a 1Ch) y, por último, las que contienen parámetros de la BIOS (1Dh a 1Fh) y que no son realmente rutinas sino tan solo punteros a tablas de datos que usa la BIOS para tener información acerca del sistema sobre el cual está corriendo y otros relacionados.

INTERRUPCIONES DOS

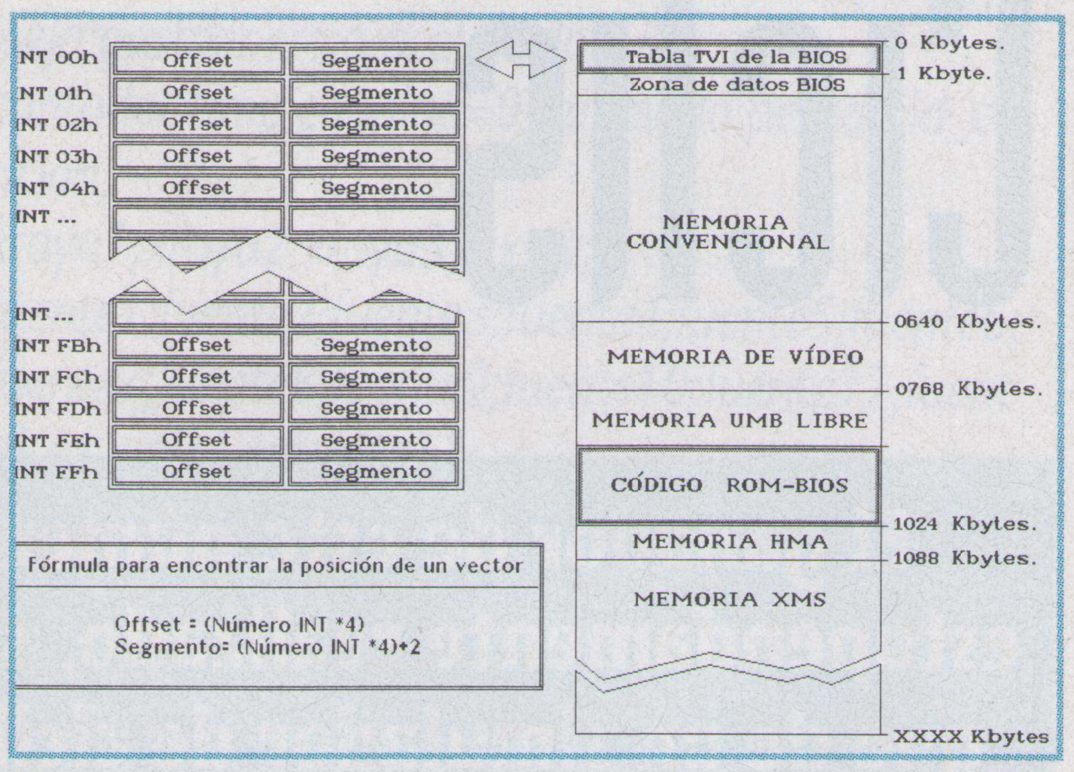
Las interrupciones del sistema Operativo DOS (MS-DOS se supone), son las que están contenidas entre la número 20h y la 3Fh (o sea, 32 interrupciones más), y algunas de las cuales poseen muchos servicios disponibles. La lista de las principales es ésta: la Int 20h, interrupción de terminación del programa, la 21h, que se usa para las peticiones de servicios, la 22h, que apunta a la dirección de terminación para cuando finalice el programa, la 23h, que posee la rutina encargada de ejecutar el proceso asociado a la pulsación de las teclas CTRL+ BREAK, la interrupción 24h, que es una rutina para el manejo de errores críticos, la 25h, que se usa para la lectura absoluta en disco, la 26h, que se usa para la escritura absoluta en disco, la 27h, que se usa para la creación de los famosos programas residentes y las restantes, las comprendidas entre las 28h y 3Fh, están reservadas para el sistema operativo y no son accesibles ni modificables.

INTERRUPCIONES DE USUARIO

Este grupo de interrupciones que son las comprendidas entre la 40h y la 7F h (un total de 64), están libres para poder ser usadas por el usuario (el programador).

COMO FUNCIONAN LAS INTERRUPCIONES

Como hemos dicho, para llamar a las interrupciones, que son rutinas ubicadas en memoria, sólo hace falta indicar su número, y no un puntero de memoria al inicio de la misma. ¿Cómo se entiende ello? pues bien, es muy fácil de entender.



ESQUEMA DE LA ESTRUCTURA Y UBICACION DE LA TVI EN MEMORIA.

En todos los sistemas hay, desde que se arranca el ordenador, un espacio de 4*256 bytes (1 kbyte) que empieza en la posición física 0 de memoria y que es una tabla con 256 punteros de memoria, donde cada 4 bytes corresponde a un valor de *OFFSET:SEGMENTO* (2+2 bytes, 2 palabras) con un puntero al inicio de una rutina o tabla de datos.
Cada número de interrupción corresponde a uno de estos punteros, y así cuando llamamos, por ejemplo, a la interrupción 21h del DOS con la instrucción **INT 21h**, lo único que hace la CPU es pasar el control de ejecución a la dirección indicada por el puntero contenido en la posición de memoria 4*21h de la tabla que hemos dicho y que empieza en el segmento 0000h (Dirección física 0), por lo que la dirección completa de la rutina se extrae de la posición de memoria: 00h:[4*21h].
A la tabla donde están todos los punteros de interrupción se la llama *Tabla de Vectores de Interrupción* (TVI), donde se considera el término vector un equivalente al de puntero. Para más detalles de cómo está diseñada esta tabla, pueden ver la fotografía número 1, donde hay un esquema de la misma.

BIBLIOGRAFÍA

- Cómo programar en ensamblador 80x86. Biblioteca técnica de Programación. Prensa Técnica.
- 8088-8086/8087 Programación ENSAMBLADOR en entorno MS-DOS, Miguel Ángel Rodríguez Roselló. Anaya Multimedia.
- Programación del 80386/387. Manual de referencia y técnicas avanzadas de programación para diseñadores de sistemas, programadores y usuarios avanzados. John H. Crawford/Patrick P. Gelsinger. Anaya Multimedia.
- La ROM-BIOS de IBM. Ray Duncan. Anaya multimedia.
- Funciones del MS-DOS, Ray Duncan. Anaya Multimedia.

JUEGOS GANADORES: 1º

Autor: Luis Rodríguez Campos

Ciclis

El ganador de este número es un juego muy original y estupendamente realizado, se trata nada más y nada menos que de un simulador de ciclismo. Tendrás que dosificar las fuerzas de tu corredor para que llegue el primero a la meta y como fondo unos paisajes estupendos, casi como una retransmisión televisiva.



No conocemos muchos simuladores de ciclismo, de hecho éste es el primero que hemos visto, una idea muy buena y además se deja jugar muy bien. Lo que si sabemos es que una empresa española está ahora mismo realizando un simulador del mismo tipo. ¿Será tan bueno como *Ciclis*? El tiempo lo dirá. Enhorabuena Luis y sigue así. A este juego le falta muy poco para ser profesional, tan poco que nos ha dicho el propio autor que ya ha conseguido vender alguno a través de su página web. En el CD metemos la demo nada más, si te gusta ya sabes.

CARACTERISTICAS

- Gráficos fotorrealistas.
- Diversas músicas originales, sonidos y voces.
- Elevada inteligencia artificial.
- Numerosas competiciones reales en las que participar, desde la vuelta a Valencia a el Tour de Francia, pasando por la Paris-Niza.
- Completo editor del perfil de cada etapa, para que las construyas a tu gusto o copies las reales del actual Tour de Francia, por ejemplo.
- Manejo de los otros ciclistas de tu equipo por tí mismo o por otro jugador.
- Numerosos gráficos adaptados al tipo de carrera.
- Escala pendientes en las que 12 km/h es mucho y descensos en los que rozarás los 100 km/h.
- Comentarios referentes a lo que pasa en carrera dichos por los ciclistas.
- Los ciclistas se comportan como en una carrera real: se ponen a rueda, lanzan ataques cuando el pelotón va a poca velocidad, se quedan si vas muy rápido, sprintan en la meta, buscan la rueda buena...
- Puedes salvar tu partida, no se corre el Tour en un día...
- Completa información en carrera de la posición de los otros ciclistas.
- Modo "historia": empiezas en aficionados y tienes que mejorar tu fuerza y tu equipo para pasar a profesionales, y finalmente ganar una Gran vuelta por etapas.
- Puedes elegir corredor, desde Sodexo, Banaka, Olarra...hasta Banesto, Kelme y Mapei.

REQUERIMIENTOS

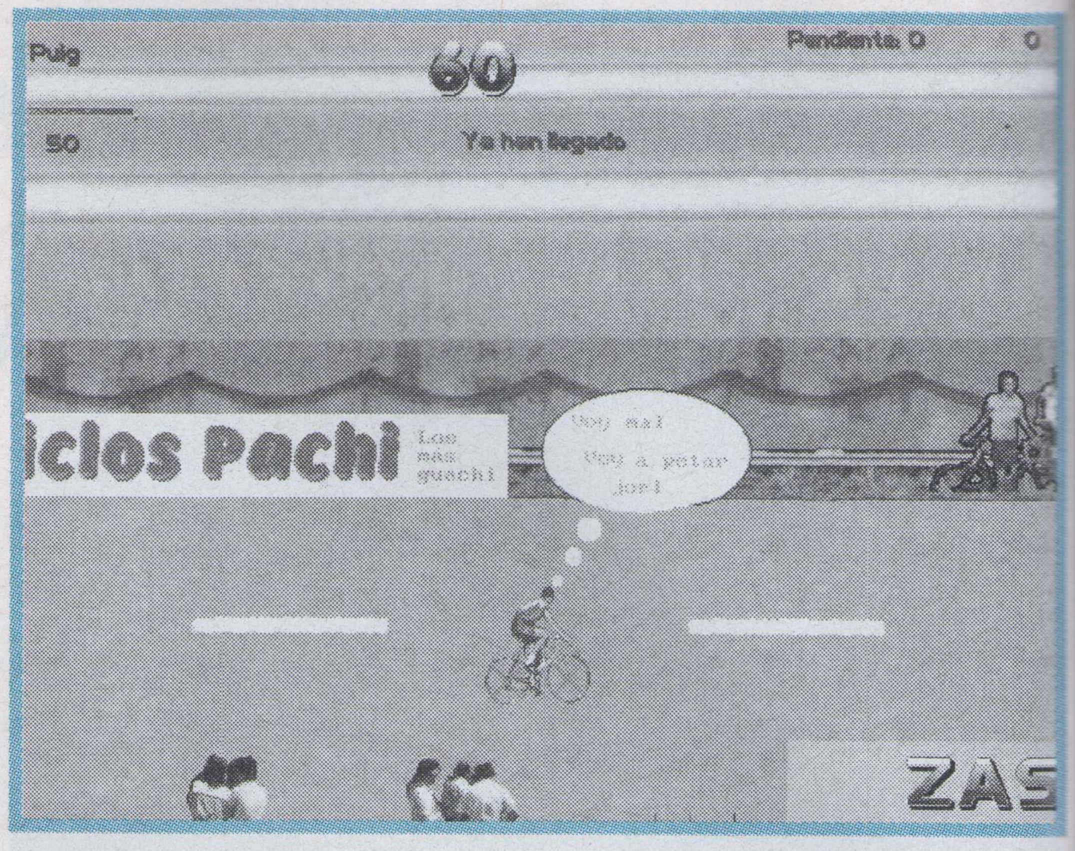
Se ha hecho con un Pentium 100,a si que puedes hacerte una idea, pero en modo de alta calidad y con 100 corredores, la cosa va un poco lentilla.

ACERCA DEL AUTOR...

En 1997 hice un *Ciclis* para Visual Basic (!) pero no llegué a hacer mas que una etapa. Luego hice un juego de estrategia que era entretenidillo, pero poco más. En verano de 1999 empecé el nuevo *Ciclis* con Div 2, un entorno de programación que facilita la inserción de gráficos, con lo que el *Ciclis* original fué retomado con fuerza. Acabado en Abril del 2000. Ahora estudio Informática de gestión en la Universidad de Córdoba.

CONTROLES

Arriba y abajo: evidentes.
Izquierda: frenar.
Derecha: Acelerar.
1,2...9: Canciones, aunque en la demo solo hay una.



0: Quitar la canción.
u: Quitar el ruido de los pedales.
F1: Disminuir volumen.
F2: Aumentar volumen.

Avanzar página: Mover la cámara a la derecha.
Fin: Volver con la cámara a tu corredor.
Suprimir: Mover la cámara a la izquierda.

C: Comer.
Alt-x: Salir del juego de golpe.
Escape: Salir de esta vuelta.
Pausa: Pausa.
Control-alt-p: Graba una captura del juego.

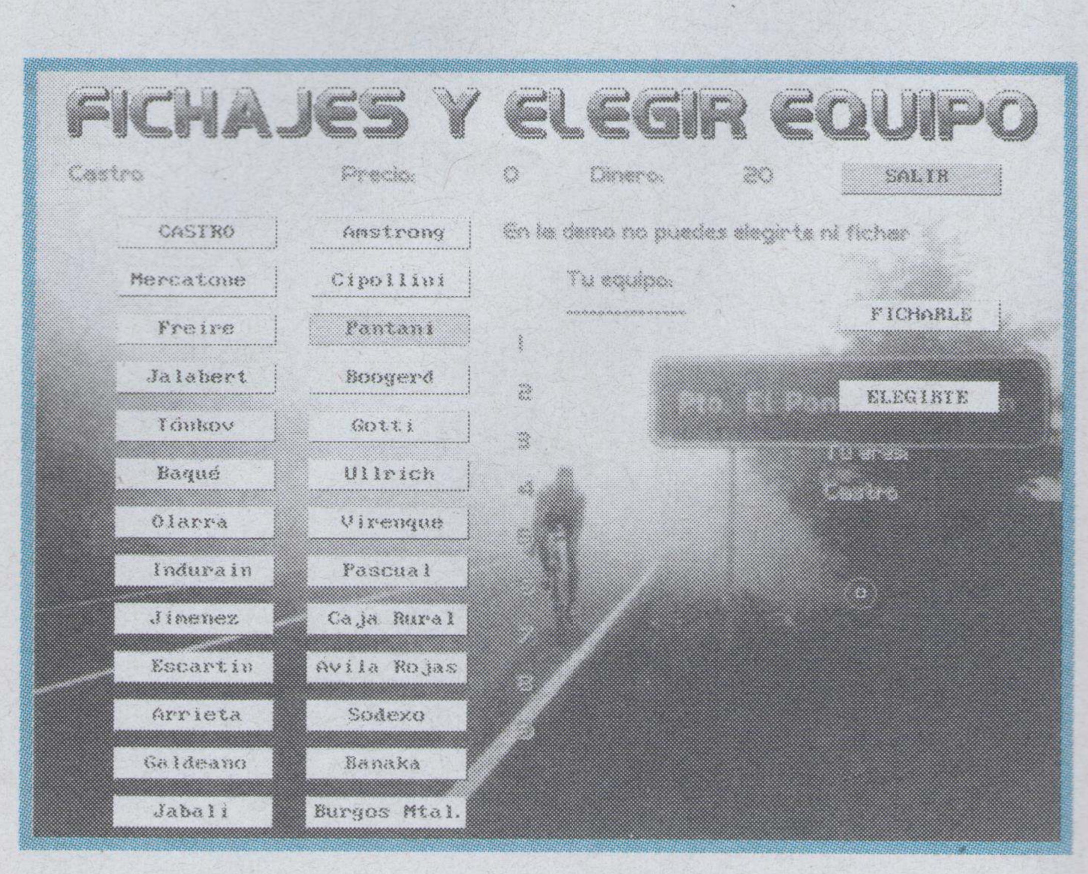
B: Aumentar brillo.
N: Disminuir brillo.
V: Rotar la intensidad de los colores primarios aleatoriamente.
Espacio: Brillo original.

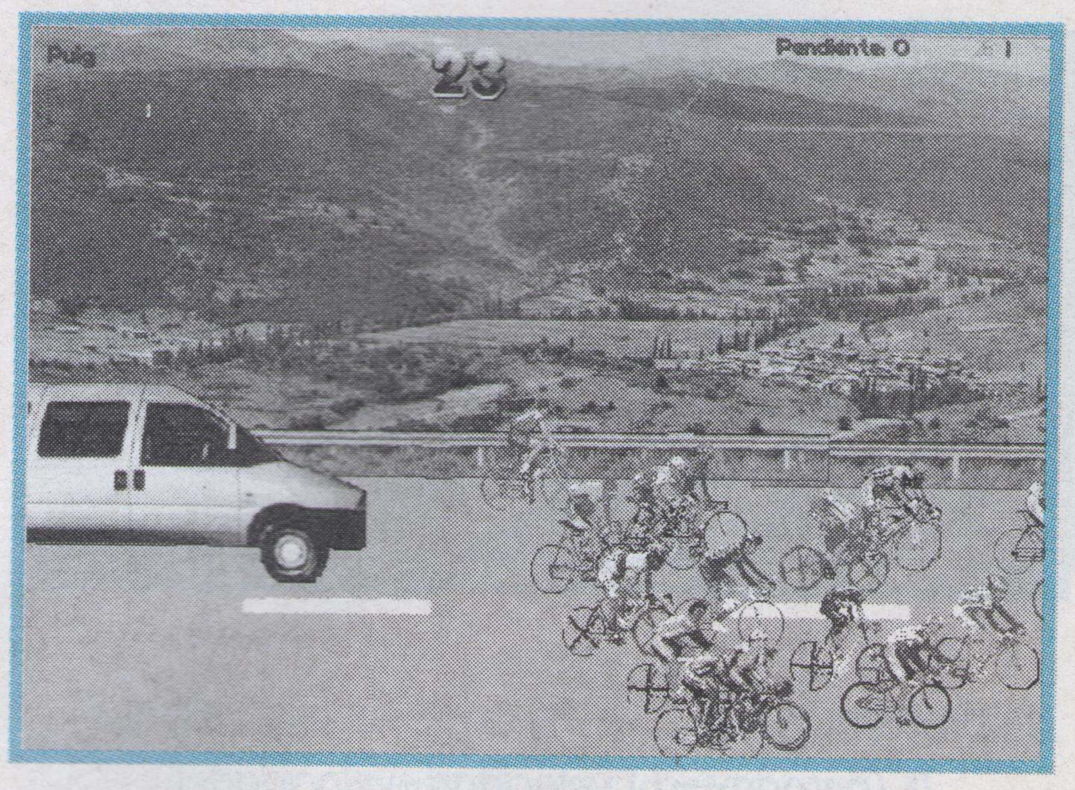
W, A, S, D: Para manejar al corredor de tu equipo que tengas seleccionado, pero como en la demo no tienes compañeros, nada. Con la versión entera se puede jugar "a dobles" si otro maneja al compañero.

Intro y Shift derecho: Cambia de compañero entre los que tengas.

INDICADORES DE PANTALLA

Arriba a la izquierda te dice el nombre de la vuelta. Justo debajo hay una barra con la energía, si llega a la raya que hay un poco más





adelante has petado y te frenas un poco. Debajo un número que indica lo que está variando la energía en estos momentos. En medio hay un número grande que indica la velocidad. A su izquierda sale la palabra 'CHUP' cuando estás chupando rueda, con lo que gastas menos energía.

Mas a la derecha te dicen la pendiente y un gráfico con el perfil de la etapa, con un punto rojo que simboliza tu posición. El numerito pequeño verde te dice las pantallas por segundo a los que te está rulando el juego, lo normal son 25. Puedes cambiarlo en 'opciones'. Aunque si tienes un ordenador del año 95 y le pones que vaya a 80, no llegará.

Arriba a la derecha viene el número que indica la comida que te queda. En la demo es solo 1 o ninguna (si ya has comido). Por último abajo del todo hay unas rayitas que representan la situación de los corredores en carrera: empiezan a la izquierda de la pantalla y la meta está al final,a la derecha. Tu eres la raya roja, los de tu equipo los verdes y el resto azules.

OPCIONES DEL JUEGO

En las pantallas del menú, cada vuelta indica con una lista de números el número de etapas que tiene esa vuelta y el tipo de etapa que es.

0: Etapa llana.
1: Media montaña.
2: Alta montaña.

En 'opciones' tipo de PC es para que los que les vaya lento desactiven algunas cosas:

0: No se mueve el fondo de la pantalla y los enemigos no chocan entre ellos.
1: Como el cero pero sí que chocan.
2: Todo activado.

Modo historia: Empiezas siendo un aficionado (sólo puedes correr las vueltas de aficionado, que están abajo en la lista de cada mes. Al darle a final de temporada, si has fichado gente y subido suficientemente tu energía, pasas al nivel 2,con lo que puedes correr con los profesionales, aunque todavía no lo seas. Las carreras de nivel 2 se reconocen porque están pintadas de morado. Si al acabar la temporada tienes más equipo y más energía, pasas a profesionales. Si no, se te mantiene acumulado para la siguiente temporada.

El editor de perfiles de etapa es bastante intuitivo: arriba incrementa la pendiente del tramo seleccionado, y abajo lo disminuye. Izquierda y derecha cambian de tramo.

Consejos
Procura chupar rueda siempre que puedas, controlando los ataques, decidiendo en que escapada te vás... Gastaté la comida (pulsar c) en el momento oportuno, cuando vayas a petar. Lanza el sprint en cuanto veas las vallas de meta.

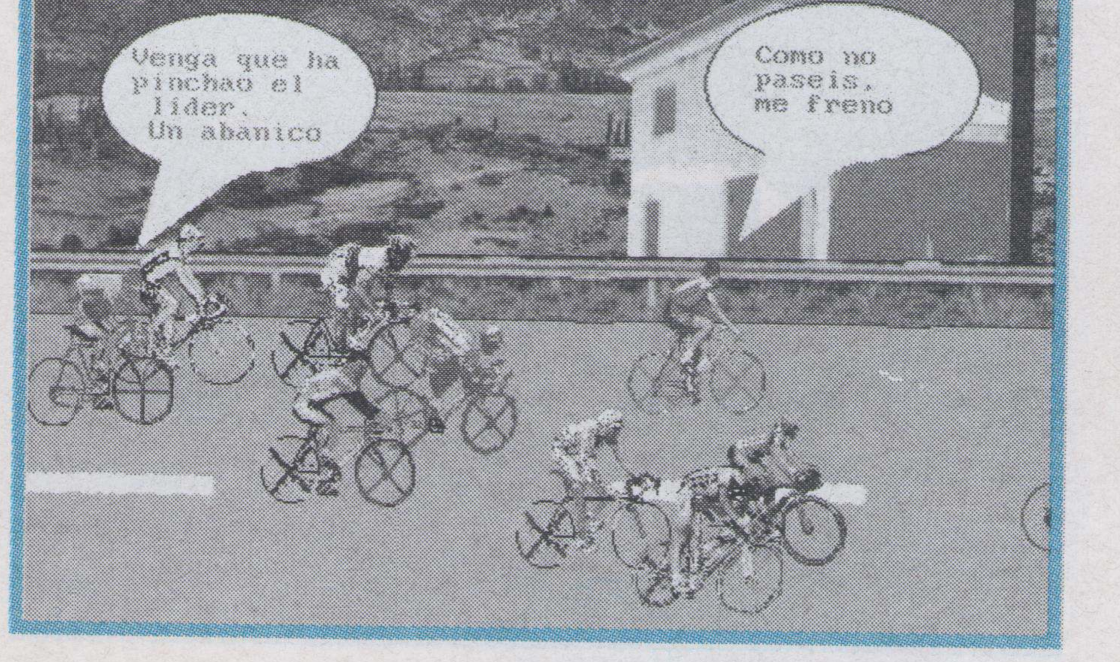
Suerte,y gracias por jugar...... .

CÓDIGO FUENTE

```
//COMPILER_OPTIONS _MAX_PROCESS=200;
//MUCHO CUIDADO
//mirar con memory_free a ver si algun punto del
programa acumula gasto de memoria
//Haciendo tablas pueden simplificarse muchas
cosas,como con enemigo
PROGRAM Ciclis;              //.1999.Luis
Rodriguez Campos.Zasoft.
//a¤adir and v>=id2.v al compilar
GLOBAL a=1;  elfondo1=312;
xrayametainicial=20000; etapa=0; //LAS 3
GRANDES
nivel_actual=3;dinero=0;comida=1;peta=25000;
pcbueno=2;
lame;lame2;lame3;lame4;lame5;lame6;lame7;la
me13;lame15;lame18;
music;c_por_equipo=1;nivel=1;tipo_etapa;
nvoltavieja;nvolta; volumen=256; s_pedal1;
dst_cabeza;dst_ultimo;
chupingvieja;  vvieja; lamevieja; llamabocata;
llamabocata2;
salir=1;histori;
sonidoquit;paclasis;choca;vpeloton=27;en_carrer
a;esc_pulsed;  borraya;
conversor[9];fichetotal;vobjetivo;quemadavieja;
xrayameta;popo;continuaractivo;string
vuelta="Leon";finished;
ponfoto;playergana;string etapaelegida;
tempoganador;tiempof;enx;eny;tp3;despierto;tem
p;
pr;distanciavieja;diego=49;dineroviejo;
fiched;fiched1;fiched2;fiched3;fiched4;
boton1;boton2;boton3;boton4;//mira a ver lo
que puedes poner private o que incluso sobre
maxim=50;
p[12];p1;p2;p3;p4;p5;p6;p7;p8;p9;p10;p11;p12
;vu1;vu2;vu3;vu4;vu5;vu6;vu7;vu41;
pendiente=0;pendientevieja;ingresos;
moral=10;distancia=0;velocidad=23;tiempo=0;y
vieja;gastoenergia;aceleracion=0;
contador;timo;chuping=0;quemada;petada;id_ca
ncion;
xraya1;xraya2;xraya3; xx;yy;
fuente1;fuente2;fuente3;fuente4;fuente5;fuente6;
fuente7;fuente8;fuente9;fuente10;fuente11;fuent
e12;fuente13;fuente14;
xfondo1;xfondo2;xfondo3;xvalla1;xvalla2;xvalla3
;xvalla4;xcamino1;xcamino2;
enemigor[101];fiche[25];
string conel1;string conel2;string conel3;string
conel4;string conel5;string conel6;string
conel7;string conel8;string conel9;
struct clasis    [101] distancias;tempo; string
nombre; end //borrar distancias?
struct clasisg    [101] tempofg;         string
nombre; end
struct clasistemp [101] temptotal;       string
nombre; end

LOCAL
ex;ey;ficher;dst;v;dstvieja;
chupi1;chupi2;chupi3;chupi4;chupi5;chupi6;chup
i7;chupi13;chupi15;chupi18;
contador2;timo2;t;tempof;tmp;
vmin2;vmax2;  pulsed;hacorrido;energy;string
name;
vmin;//velocidad minima a la que un enemigo
lanza un ataque
vmax;//       "     maxima          "
//redmin;//cuantos km/h reduce (como
minimo)cada 10s en un ataque
redmax;//          "              maximo         "
//tipmin;//cuanto tiempo tarda como minimo en
dar un ataque,cuando va a velocidad jose
tipmax;//          "         maximo         "

BEGIN
  set_fps(25,1);//me veo obligado a poner 1
  set_mode(m640x480);
  music=rand(1,3);
  if(music==1)
load_song("..\ciclis\ciclis1.xm",1);end
  if(music==2)
load_song("..\ciclis\ciclis2.xm",1);end
  if(music==3)
load_song("..\ciclis\ciclis3.xm",1);end
  song(id_cancion);
```



Juegos ganadores: 1º

```
fuente1=load_fnt("..\ciclis\ciclis1.fnt");
fuente2=load_fnt("..\ciclis\ciclis2.fnt");
fuente3=load_fnt("..\ciclis\ciclis3.fnt");
fuente4=load_fnt("..\ciclis\ciclis4.fnt");
fuente5=load_fnt("..\ciclis\ciclis5.fnt");
fuente6=load_fnt("..\ciclis\ciclis6.fnt");
fuente7=load_fnt("..\ciclis\ciclis7.fnt");
fuente8=load_fnt("..\ciclis\ciclis8.fnt");
fuente9=load_fnt("..\ciclis\ciclis9.fnt");
fuente10=load_fnt("..\ciclis\ciclis10.fnt");
fuente11=load_fnt("..\ciclis\ciclis11.fnt");
fuente12=load_fnt("..\ciclis\ciclis12.fnt");
fuente13=load_fnt("..\ciclis\ciclis13.fnt");
fuente14=load_fnt("..\ciclis\ciclis14.fnt");
// dump_type=partial_dump;//quita esto si usas scrolls
// restore_type=partial_restore;

intro();
frame;
END
//——————————————————————
//———————————PRINCIPAL————
//——————————————————————

PROCESS principal()
PRIVATE rapid; id1;id2;id3; pop; esc; n=1; sprint; pe;
BEGIN id1=raya1();id2=raya2();id3=raya3();
graph=diego; y=300; x=320; xx=x;

s_pedal1=load_wav("..\ciclis\voces\pedal1.wav",1);
    LOOP //borraya=memory_free();
    if(pr==1)pr=0;end //la segunda vez que entre aqui
    IF (pr==2)pr=1;perfilin(); mouse.graph=327;
pendiente=0; angle=0; graph=diego; esc=0;

sonidoquit=sound(s_pedal1,volumen/4,256);player
gana=0; esc_pulsed=0;sprint=0;
    dst_cabeza=0;fade(90,90,90,100);
    for(n=1;n<=fichetotal;n++)
enemigor[conversor[n]].ficher=1; end
    if(histori==1 and nvoltavieja>=nvolta and
etapa==1 and salir==1)etapa=0; finished=0;
esc_pulsed=1; calculos(); end
    if(histori==1 and nivel!=nivel_actual)
etapa=0; finished=0; esc_pulsed=1; calculos();
end
    if(esc_pulsed!=1)/*nvoltavieja=nvolta;*/salir=0;
end //para que solo se asigne nvoltavieja a las que
realmente se han corrido
    END
    if(key(_del))x=x+75;end if(key(_pgdn))x=x-
75;end if(key(_end))x=320;end
    if (xrayameta<5000)x=320;end
    if (pcbueno>1)angle=pendiente*1000;end

    if(pendiente>0) pe=3; else pe=6; end
    gastoenergia=velocidad-
35+rapid+pe*pendiente-chuping;//O sea,en
principio gatoenergia=velocidad,lo demas son
a¤adidos.
    if (velocidad<10 and gastoenergia>-
10)gastoenergia=-10;end//para que en pendientes
del 14% no gaste energia hasta parado.

    if (velocidad>35) rapid=velocidad-35; else
rapid=0; end//esto es que a partir de 35km/h
gastas aun mas

    if (petada==0)
quemada=quemada+gastoenergia;end
//  if (quemada<quemadavieja) quemada-=10;
end //para que disminuya mas rapido,si no tarda
mucho. Y no depende de la velocidad porque te
recuperas lo mismo a 25 km/h que a 0 Km/h
    if (quemada<1) quemada=0; end
    if (petada==1)quemada=quemada-30;end
    if (quemada<1) quemada=0; end //lo repito
para que no de problemas
//  varquemada=quemada-quemadavieja;
//  quemadavieja=quemada;

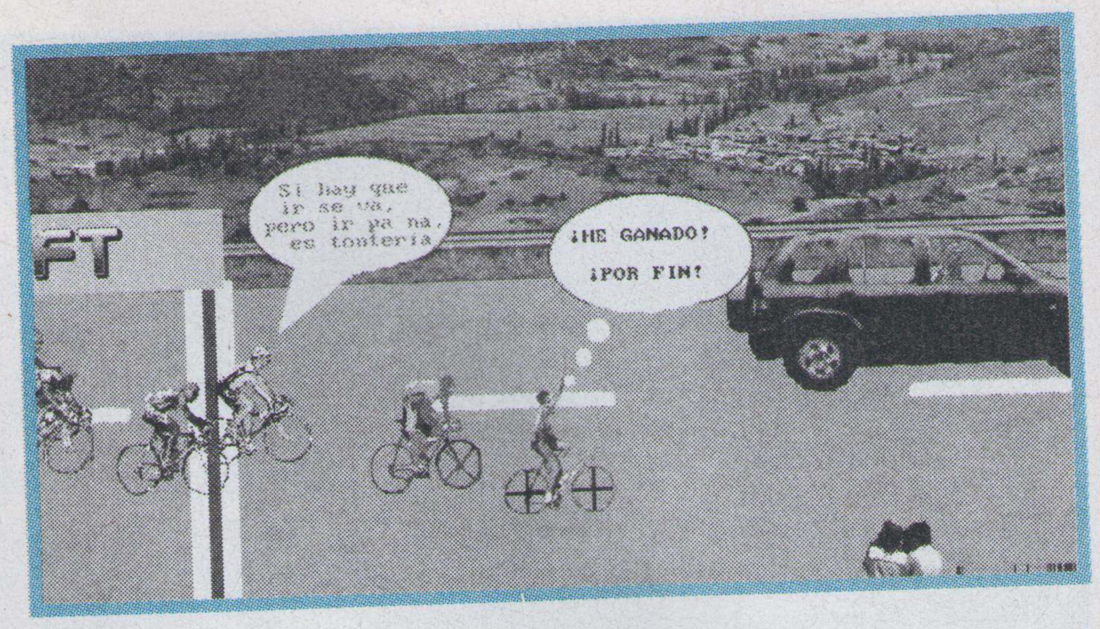
//————————————————Cambio de
Velocidad————————————

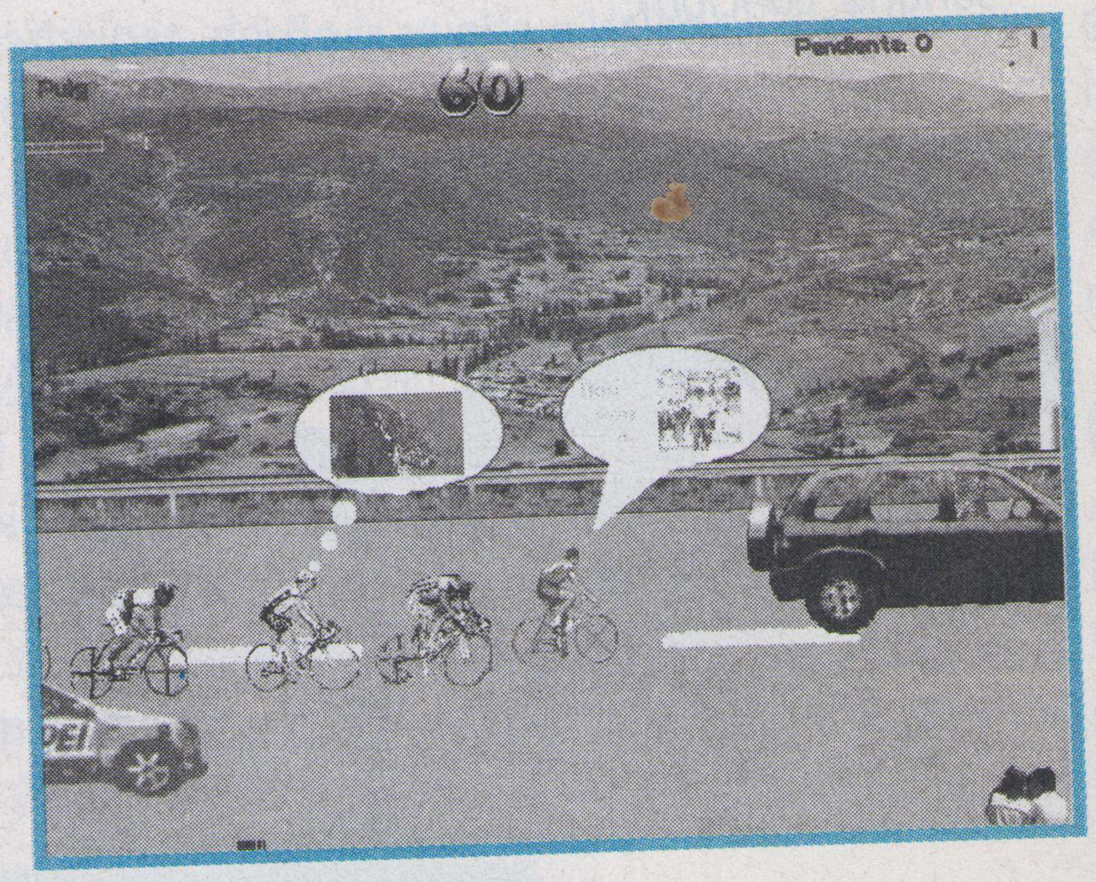
/*
    if (key(_right))vobjetivo++;end if
(key(_left))vobjetivo—;end
    IF (moral mod 3==0) //antes era 10,no 3,ver si
da problemas
    IF (petada==1) velocidad=velocidad-1 ;
    ELSE if (velocidad<vobjetivo)/*
aceleracion=aceleracion+1;*/velocidad=velocidad+
1;end
    END
    if (velocidad>vobjetivo)
velocidad=velocidad-1;end
        vobjetivo=velocidad;
            //DESHACERSE DE ACELERACION
//      if (key(_right)==0 and aceleracion>0)
aceleracion=aceleracion-5;end
//      if (aceleracion<0) aceleracion=0;end
    END
*/
    if(petada==1)velocidad—;end
    if(key(_right) and petada==0)velocidad++;
end
    if(key(_left)) velocidad—; end
//——————————————————————

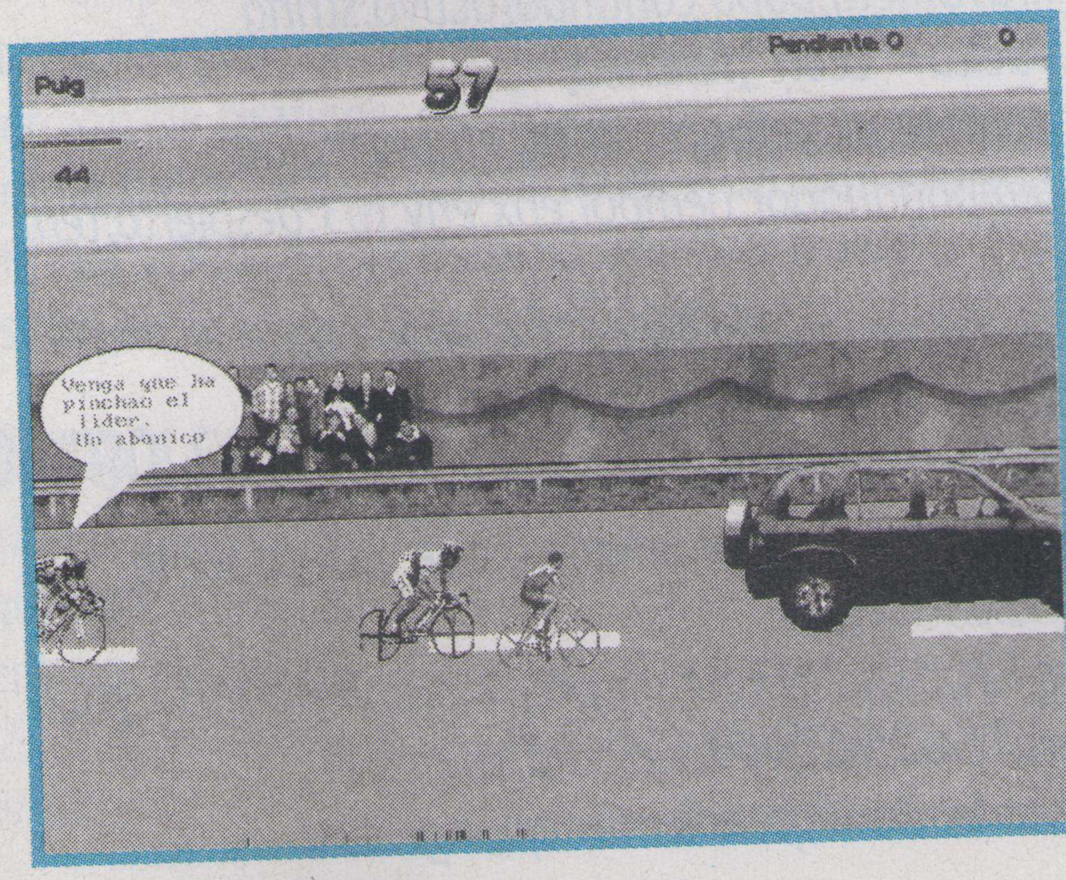
    if (velocidad<1) velocidad=0;end
    maxim=60-3*pendiente; //antes era
maxim=50-2*pendiente;
    if (xrayameta<5000) sprint=1; end
    if (velocidad>maxim and xrayameta>5000)
velocidad=maxim;end
    if (velocidad>maxim+15)
velocidad=maxim+15; end
    if (pendiente==0 and velocidad>maxim)
velocidad=maxim; end

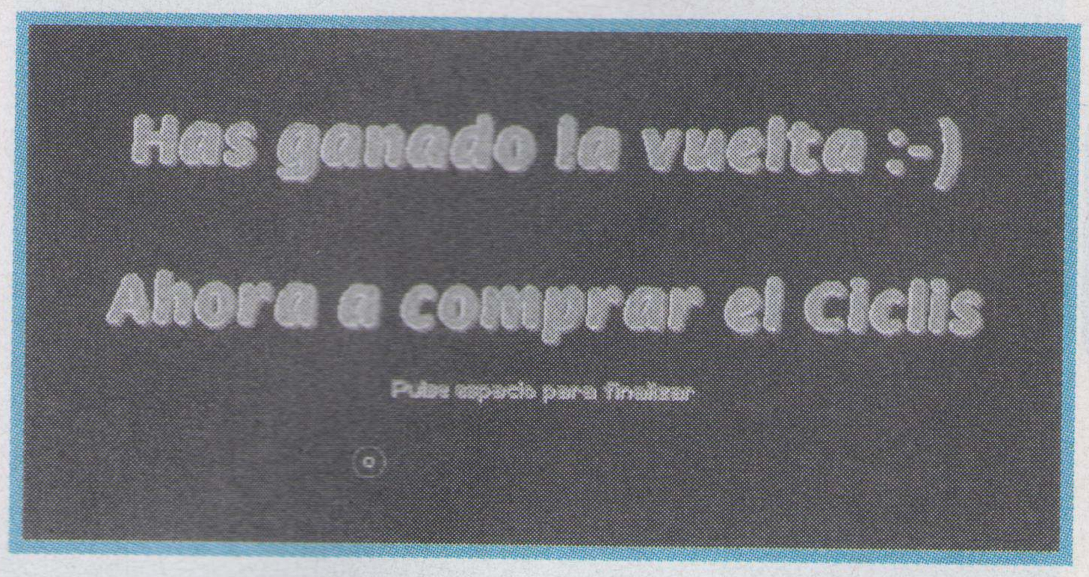
    //los cambios de personaje
    if (velocidad>0)contador=contador+2;
timo=200/velocidad;end
    IF (contador>timo and graph==diego-1)
graph=diego; contador=0; END
    IF (contador>timo and graph==diego)
graph=diego-1; contador=0; END
    IF (diego==49 or diego==46)
    if (key(_right)    and graph>47)graph=46;
diego=46; end
    if (not key(_right)and graph<47)graph=49;
diego=49; end
    if (playergana==1) graph=47; end
    END

    z=-(y);
    //——————————————————
PENDIENTE————————————
```









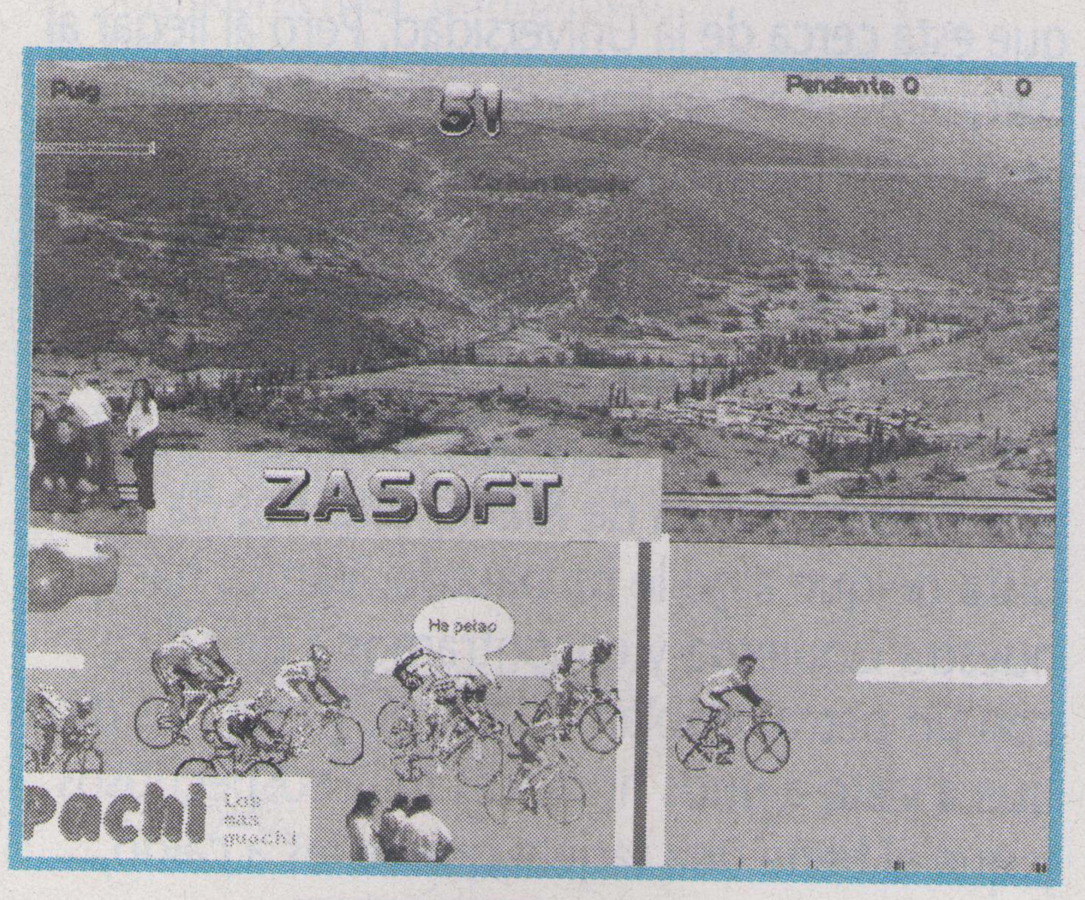

```
—
    pendientevieja=pendiente;
  popo=xrayametainicial-xrayameta;
    //Lo de la pendiente es asi:

    IF (moral mod 25==0 and pcbueno<2)
  pop=xrayametainicial/12;
        nvoltavieja=nvolta;
      IF (popo>0 and popo<pop       )
  pendiente=p[1];end
      IF (popo>pop and popo<pop*2   )
  pendiente=p[2];end
      IF (popo>pop*2 and popo<pop*3 )
  pendiente=p[3];end
      IF (popo>pop*3 and popo<pop*4 )
  pendiente=p[4];end
      IF (popo>pop*4 and popo<pop*5 )
  pendiente=p[5];end
      IF (popo>pop*5 and popo<pop*6 )
  pendiente=p[6];end
      IF (popo>pop*6 and popo<pop*7 )
  pendiente=p[7];end
      IF (popo>pop*7 and popo<pop*8 )
  pendiente=p[8];end
      IF (popo>pop*8 and popo<pop*9 )
  pendiente=p[9];end
      IF (popo>pop*9 and popo<pop*10 )
  pendiente=p[10];end
      IF (popo>pop*10 and popo<pop*11)
  pendiente=p[11];end
      IF (popo>pop*11 and popo<pop*12)
  pendiente=p[12];end
    END
    IF (moral mod 25==0 and pcbueno==2)
  pop=xrayametainicial/12;
      nvoltavieja=nvolta;
    IF (popo>0 and popo<pop         and
  pendiente!=p[1])if(pendiente>p[1])pendiente—;
  else pendiente++;end END
    IF (popo>pop and popo<pop*2    and
  pendiente!=p[2])if(pendiente>p[2])pendiente—;
  else pendiente++;end END
    IF (popo>pop*2 and popo<pop*3   and
  pendiente!=p[3])if(pendiente>p[3])pendiente—;
  else pendiente++;end END
    IF (popo>pop*3 and popo<pop*4   and
  pendiente!=p[4])if(pendiente>p[4])pendiente—;
  else pendiente++;end END
    IF (popo>pop*4 and popo<pop*5   and
  pendiente!=p[5])if(pendiente>p[5])pendiente—;
  else pendiente++;end END
    IF (popo>pop*5 and popo<pop*6   and
  pendiente!=p[6])if(pendiente>p[6])pendiente—;
  else pendiente++;end END
    IF (popo>pop*6 and popo<pop*7   and
  pendiente!=p[7])if(pendiente>p[7])pendiente—;
  else pendiente++;end END
    IF (popo>pop*7 and popo<pop*8   and
  pendiente!=p[8])if(pendiente>p[8])pendiente—;
  else pendiente++;end END
    IF (popo>pop*8 and popo<pop*9   and
  pendiente!=p[9])if(pendiente>p[9])pendiente—;
  else pendiente++;end END
    IF (popo>pop*9 and popo<pop*10  and
  pendiente!=p[10])if(pendiente>p[10])pendiente
  —; else pendiente++;end END
    IF (popo>pop*10 and popo<pop*11 and
  pendiente!=p[11])if(pendiente>p[11])pendiente
  —; else pendiente++;end END
    IF (popo>pop*11 and popo<pop*12 and
  pendiente!=p[12])if(pendiente>p[12])pendiente
  —; else pendiente++;end END
    END
    if (pendiente!=0 and pcbueno>1)
  signal(id1,s_sleep);signal(id2,s_sleep);signal(id3,s
  _sleep);
    else
  signal(id1,s_wakeup);signal(id2,s_wakeup);signal
  (id3,s_wakeup);end
  //   if (key(_l)) pendiente—;end if (key(_o))
  pendiente++;end
  //————————————————————————————————————————

    tiempo=1+timer/100;

    distanciavieja=distancia;//esto ha de ir antes
  que la linea siguiente para que funcione chupi1
    distancia=distancia+10*velocidad;
    moral=moral+1;//la moral solo es un contador
  que suma 25 por segundo

    //ahora los controles de parriba y pabajo
    IF (NOT fading AND
  key(_v))fade(rand(70,130), rand(70,130),
  rand(70,130),1);END
    if (key(_space)) fade(90,90,90,1); end
    if (key(_b)) fade(110,110,110,1); end
    if (key(_n)) fade(60,60,60,1); end
```




```
  //   if (key(_esc)) esc=tiempo; write
  (fuente4,320,100,4,"PRESIONE ENTER PARA
  ABANDONAR"); end
    if (key(_esc)/* and tiempo<esc+3*/)
  en_carrera=0;  etapa=0;  finished=0;
  esc_pulsed=1; stop_sound(sonidoquit); calculos();
  end

     if (yy!=y) y=yy; end
     choca = collision(TYPE enemigo); //puede que
  de problemas si colisiona al empezar,antes de
  que haya cargado los proceso enemigos
     IF (choca<>0)
  //     if (choca.v<velocidad-20 and tiempo>10)
  //caidas
  //
  chupi1=0;chupi2=0;chupi3=0;chupi4=0;chupi5=
  0;chupi6=0;chupi7=0;chupi13=0;chupi15=0;chu
  pi18=0;
  //       velocidad=0; choca.v=1;
  //    end
     if(y*25<choca.y+500 and y*25>choca.y-
  1)y=y+4;end
     if(y*25>choca.y-500 and y*25<choca.y)y=y-
  4;end //copiar este concepto al player
    END
  //    y=yy;
    if (key(_up)) y=y-4; end     if (y<265)
  y=265;end
    if (key(_down)) y=y+4; end   if (y>440)
  y=440;end
    yy=y;xx=x;

  //——————————————llamadas a
  bocatas——————————————
   if(velocidad<vvieja and chuping==0 and
  chupingvieja>0) llamabocata=411; end
   if(lame<lamevieja and lame<3 and
  velocidad>maxim-5)llamabocata=406;end
   chupingvieja=chuping;  vvieja=velocidad;
  lamevieja=lame;

  lame=0;lame2=0;lame3=0;lame4=0;lame5=0;la
  me6=0;lame7=0;lame13=0;lame15=0;lame18=
  0;
   chuping=0;//no se puede poner een el proceso
  enemigos pues lo leen muchos y entonces alguno
  se lo asigna siempre


   if
  (distancia>dst_cabeza)dst_cabeza=distancia;end
   if
  (distancia<dst_ultimo)dst_ultimo=distancia;end
   dst_ultimo=1000000;

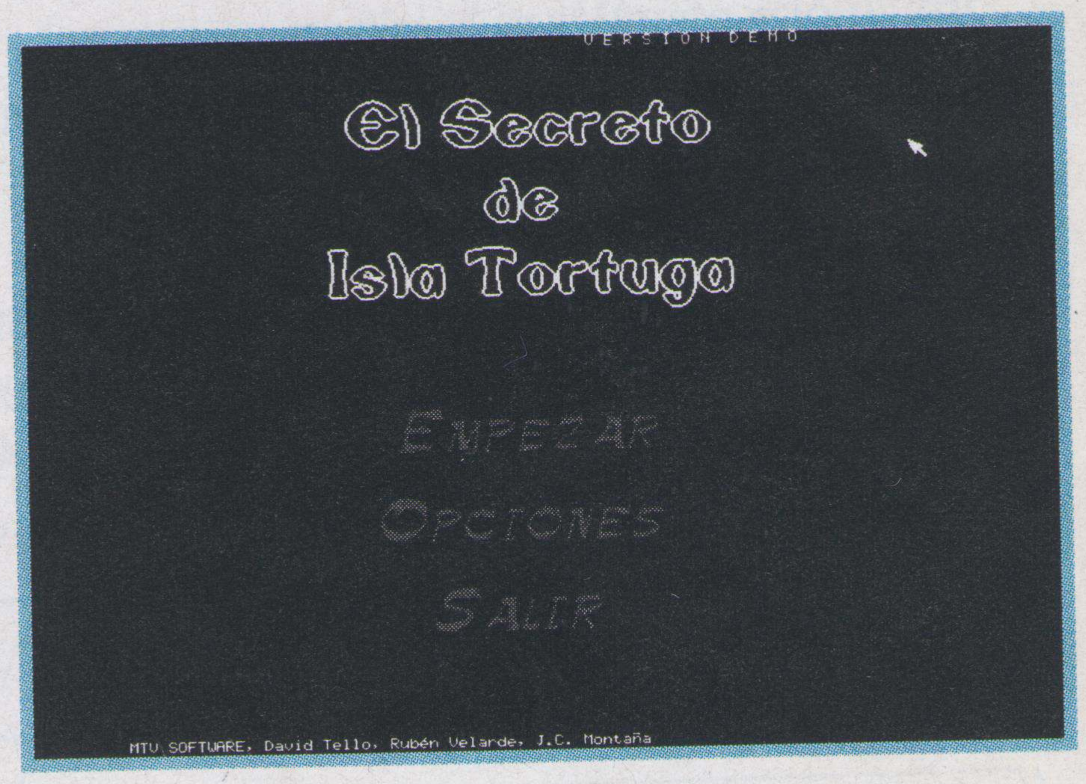
  FRAME; END END

  //————————————————————————————————————————
```

Autores: Juan Carlos Montaña, David Tello y Rubén Velarde

El Secreto de Isla Tortuga

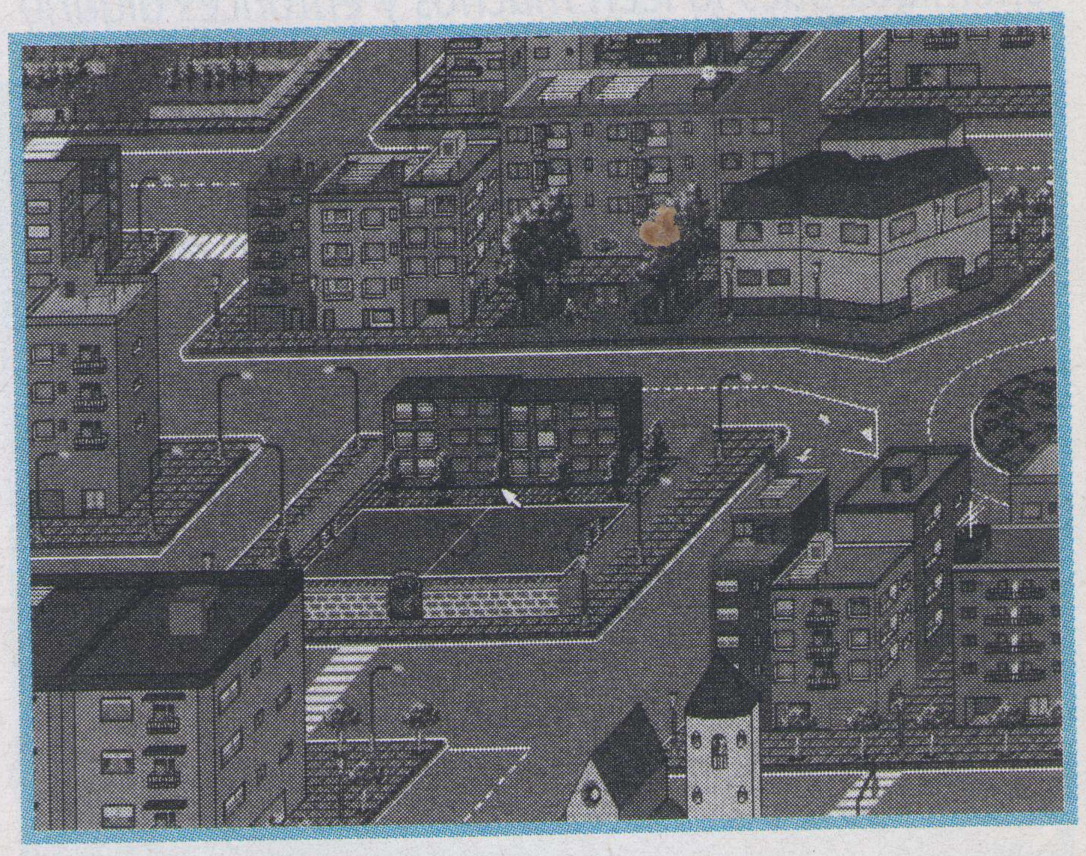
El segundo premio de nuestro concurso es una aventura gráfica en el que tendremos que estrujarnos un poco el cerebro para encontrar la solución de los problemas que se nos irán planteando, aunque los autores nos han hecho llegar una guía para aquellos que se atoren más de la cuenta.



Como dicen los autores de El secreto de la Isla de la Tortuga, no es muy usual ver aventuras gráficas hechas en DIV, aunque nosotros recordamos una en la que el protagonista tenía que meterse en un estadio de fútbol. Éste debe ser el segundo o tercer juego de este género que nos ha llegado a la redacción y merece un segundo premio, aunque esté incompleta. Quizás su código anime a otros a realizar este tipo de juegos. Nuestra enhorabuena a los ganadores.

LA OPINION DE LOS PROGRAMADORES

Nuestra opinión sobre el juego es que no es un juego muy corriente en Div, además no sabíamos nada de programar, así que nos lanzamos a hacer la aventura, que como se puede comprobar es un lío de comandos y demás, es casi imposible incluso para nosotros saber como lo hicimos.



DE DONDE PARTIO LA IDEA?

La idea partió de la base que veíamos que la gente solo hacia juegos de coches, lucha, y lo que más, de arcade. Empezamos pues a intentar ver si era posible hacer que el ratón detectase objetos, y que los pudiéramos coger, y una vez que lo conseguimos nos metimos en el argumento de la historia, que no está muy elaborado pero es válido.

DIFICULTADES

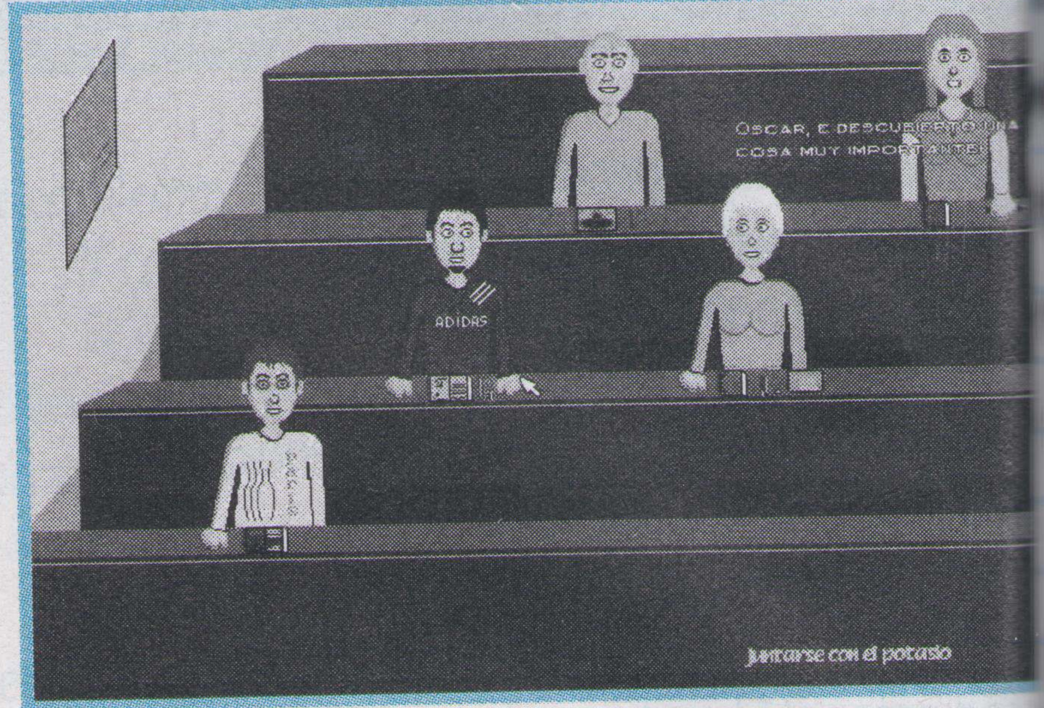
Las mayores dificultades fueron los dibujos, ya que entre nosotros tres solo hay dos que dibujan lo suficientemente bien, y entre los dos, cada uno tenía una forma diferente, por lo que nos tuvimos que adecuar al juego. Muchos dibujos los retocamos y los retocamos porque no había forma que quedaran bien, y aún así hay muchos fallos.

¿QUÉ ES LO MEJOR DE LA AVENTURA?

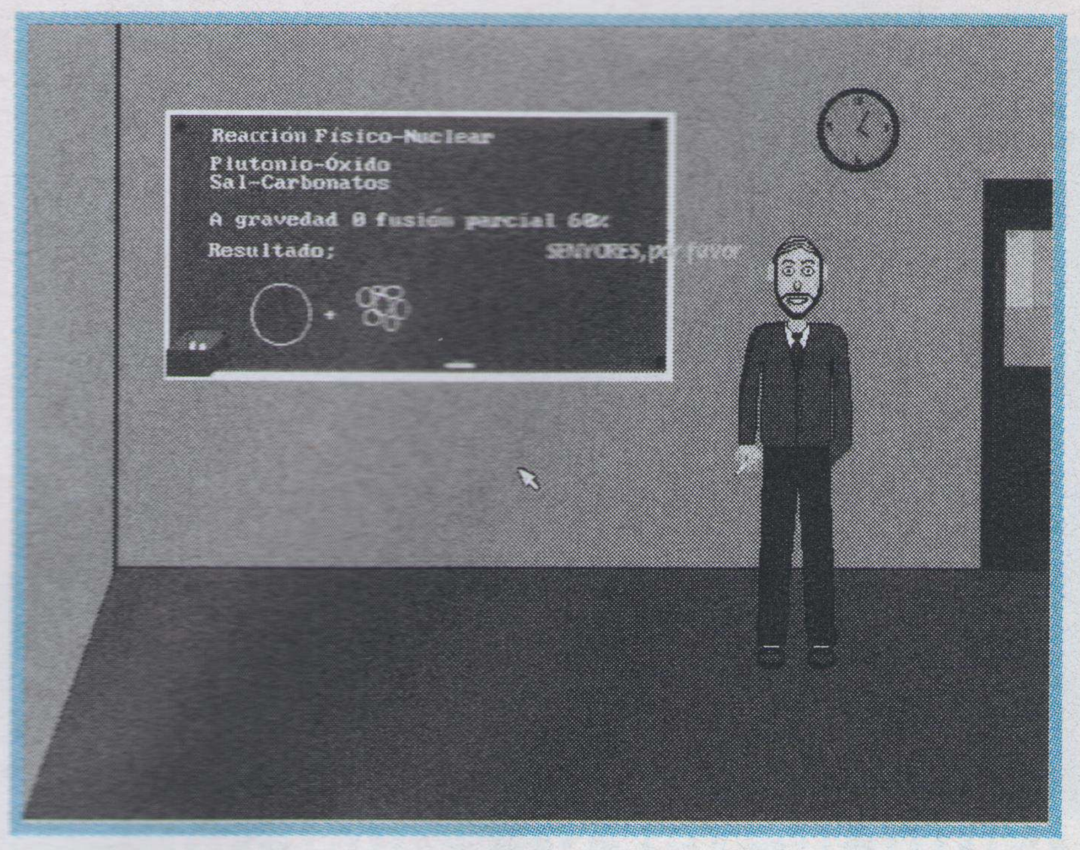
Para nosotros lo mejor es el mapa, porque fue muy difícil crearlo, debido a que para empezar no sabíamos nada de Div, nos guiábamos por lo que otras personas hacían, y no nos servía, porque no lo entendíamos, así que tuvimos que ir probando, hasta que un día acertamos, y muy bien (para nosotros).

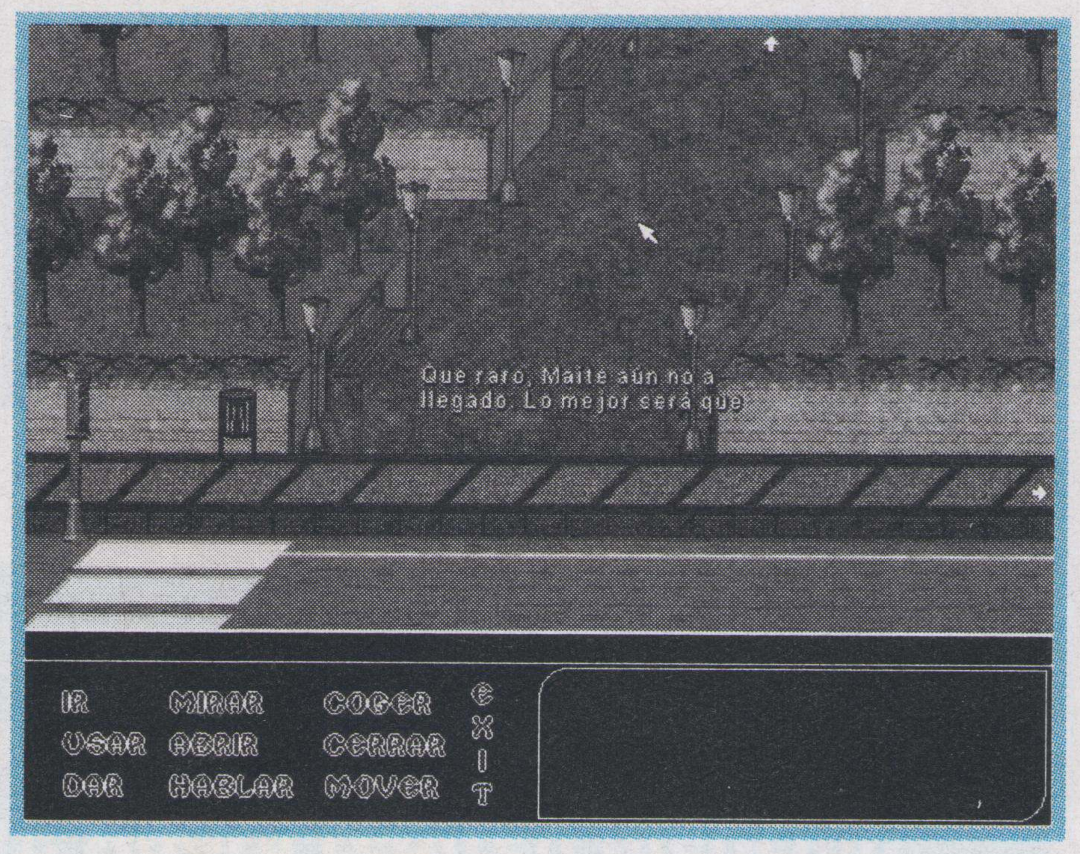
SOLUCIÓN DEL JUEGO

Nuestra amiga Maíte ha descubierto una cosa muy importante, por eso nos cita en el parque que está cerca de la Universidad. Pero al llegar al parque no la encontramos, y por eso decidimos ver si encontramos algún rastro de ella.
En la entrada del parque, podemos ver una papelera, en la cuál al mirar vemos que hay un chicle dentro. Seguimos adelante, y vamos a la fuente que hay en el parque, allí encontramos una pulsera que es de Maíte, esto nos da la pista de que algo le ha pasado. Vamos hacia nuestra izquierda y encontramos a un chico con un perro. Hablamos con él, y este nos dice que a visto a unos hombres muy sospechosos, y que el tiene una tarjeta que se les a caído,



nuestro objetivo es conseguir esa tarjeta, pero el no nos la quiere dar, hasta que después de insistir bastante, nos dice que nos la dará si le conseguimos una pelota que se le ha colado en un árbol. Así pues, solo hemos de buscar e árbol, que está a nuestra izquierda, pero al intentar coger la pelota no llegamos, necesitamos algo para alcanzarla.
Vamos hasta la fuente de nuevo, y ahora cogemos el camino de la derecha, en primer lugar encontramos a un jardinero, que está regando como cada día, y seguimos hacia la derecha hasta encontrar una caseta, abrimos la puerta, y entramos.
Dentro de ésta encontramos una escalera, la cuál cogemos, y volvemos al sitio donde estaba el árbol, usamos la escalera con el árbo y cogemos por fin la pelota. Ahora se la damo a el chico, y este nos da la ansiada tarjeta.
Ahora ya podemos irnos del parque. En el mapa, tenemos 3 sitios donde pulsar, uno es e parque otra vez, el otro la Universidad, y el último nuestra casa. Podemos ir a nuestra cas y allí subimos hasta nuestro piso. Intentamos abrir la puerta, pero no podemos porque está cerrada con llave, por eso miramos debajo de la esterilla, allí encontramos la llave, la usamos con la puerta y entramos, vamos por la casa, y no encontramos nada que nos pueda servir.
Nos vamos a la calle otra vez, y vamos a la Universidad. Allí podemos ir a ver al director, y nos interesa bastante entrar en su despacho, para eso tenemos que ir al lavabo de chicos, y allí usar el chicle con la pica, pero éste no se queda enganchado, así que tendremos que volver al parque y usar el chicle en la resina que suelta el árbol. Una vez lo hayamos hecho, usamos otra vez el chicle con la pica, y accionamos el grifo. Lo siguiente será una inundación enorme, por eso el director saldrá de su despacho y por fin podremos entrar. Y aquí se acaba esta pequeña demo de ejemplo.





CODIGO FUENTE

```
program Esdit;
global
    fichero,fichero2,fichero3;
fuente,fuente2,fuente3,fuente4,fuente5,fuente6,f
uente7,fuente8,fuente9,fuente10;
    creditos,credito2;
    titulo,titulo2,titulo3,titulo4,menu1,menu2;
    opcion=0;
    pantalla=0;
    usar_objeto;
    dar_objeto;
    hablar_con_persona=1;
    x1,y1;
    puerta;
    felpudo;
    puerta_abierta_casa;
    usar=0;
    numero=0; //sirve para el texto de las
personas
    conversacion;
    pide_balon=0;
    boca_persona1;
    boca_persona2;
    boca_director;
    video_inundacio=0;
    primera_vez_despacho=0;
    stop_saludo=0;
    activa_ojos=0;
    pa=2;
    acceso_denegado_pasillo_inundado=0;
    despacho_con_luz=0;
    acess=0;
    credits;
    mapa=0; //detecta_caminos con
start_scroll(0,...)

    //-objetos juego- que se pueden coger
    chicle=2;
    pulsera=0;
    tarjeta=0;
    escalera=0;
    pelota=0;
    llave=2;
    chicle_pegajoso=0;
    no_chicle_pegajoso=0;
    desague_taponado=0;
    grifo_encendido=0;
    //———-

    i=0; //variable de tiempo (contador)

    //objetos para inventario
    chicle_inventario;
    pulsera_inventario;
    escalera_inventario;
    pelota_inventario;
    tarjeta_inventario;
    llave_inventario;

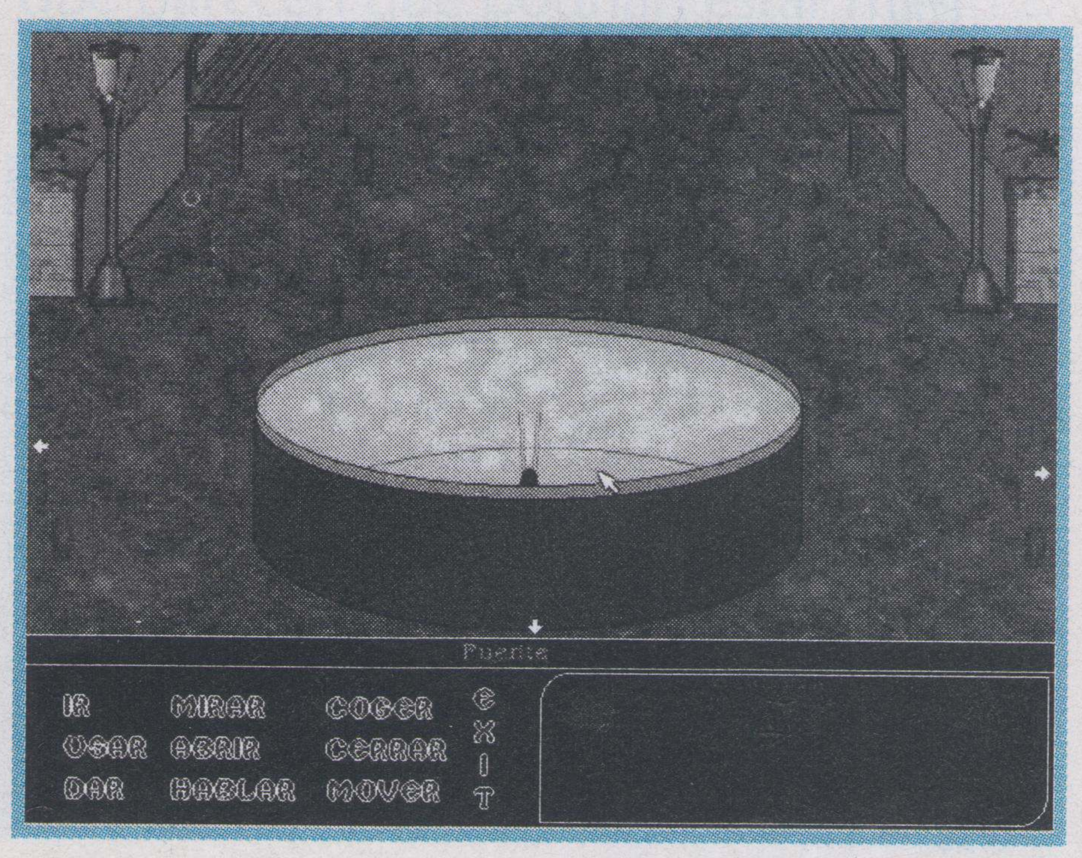
    //——————————
    papelera_pantalla=0;
    chicle_pantalla=0;
    pulsera_pantalla=0;
    puerta_pantalla=0;
    escalera_pantalla=0;
    cortacesped_pantalla=0;
    cubo_pantalla=0;
    pelota_pantalla=0;
    arbol_pantalla=0;
    perro_pantalla=0;
    font_pantalla=0;
    lata_pantalla=0;
    saco_pantalla=0;
    escoba_pantalla=0;
    biblioteca_pantalla=0;
    biblioteca2_pantalla=0;
    felput_pantalla=0;
    llave_pantalla=0;
    puerta_casa_pantalla=0;
    puerta_habitacion_pantalla=0;
    sofa_comerdor_pantalla=0;
    televisor_pantalla=0;
    boton_pantalla=0;
    puertalavhom_pantalla=0;
    lavhomdentro_pantalla=0;
    grifo_lavabo_pantalla=0;
    pica_lavabo_pantalla=0;
    resina_arbol_pantalla;
    persona1=0;
    persona2=0;
    puerta_despac_pantalla=0;
    puer_desp_pantalla=0;
    videoo=0;
    viven=0;
    empezar;
    ha=0;
    ton=0;

begin
    set_mode(m640x480);
    fichero=load_fpg("e:\esdit\fpg\esdit.fpg");
    fichero3=load_fpg("e:\esdit\fpg\movis.fpg");
    fuente=load_fnt("e:\esdit\fnt\accion.fnt");
    fuente2=load_fnt("e:\esdit\fnt\accion2.fnt");
    fuente3=load_fnt("e:\esdit\fnt\barra.fnt");
    fuente4=load_fnt("e:\esdit\fnt\habla.fnt");
    fuente5=load_fnt("e:\esdit\fnt\habla2.fnt");
    fuente6=load_fnt("e:\esdit\fnt\texto.fnt");
    fuente7=load_fnt("e:\esdit\fnt\texto2.fnt");
    fuente8=load_fnt("e:\esdit\fnt\habla4.fnt");
    fuente9=load_fnt("e:\esdit\fnt\habla5.fnt");
    fuente10=load_fnt("e:\esdit\fnt\habla7.fnt");
    titulo=load_fnt("e:\esdit\fnt\titulo.fnt");
    titulo2=load_fnt("e:\esdit\fnt\titulo2.fnt");
    titulo3=load_fnt("e:\esdit\fnt\continua.fnt");
    titulo4=load_fnt("e:\esdit\fnt\titulo3.fnt");
    menu1=load_fnt("e:\esdit\fnt\menu.fnt");
    menu2=load_fnt("e:\esdit\fnt\menu2.fnt");
    creditos=load_fnt("e:\esdit\fnt\creditos.fnt");
    credito2=load_fnt("e:\esdit\fnt\credito2.fnt");
    fade_on();

    if(hablar_con_persona==0 or
hablar_con_persona==1)
        raton();
    end
    menu();

    loop
        delete_text(all_text);
        clear_screen();

        poner_fondo_pantalla();
        if(videoo==1)
          if (hablar_con_persona==0)
            saber_donde_se_pulsa();
            objetos_inventario();
            detecta_objetos_inventario();
detectarse_objetos_inventario_entre_si();
            poner_letras_accion();
```






```
                if (pantalla>0)
                    put(fichero,202,320,240);
                end
            end
        end
        personajes();
        frame;
    end
end


process objeto(graph,size,x,y);
begin
    xput(fichero,graph,x,y,0,size,0,0);
end


process poner_letras_accion();
begin
    if (mouse.x>31 and mouse.x<49 and
mouse.y>407
        and mouse.y<420)
        write(fuente2,32,410,4,"IR");
        if (mouse.left)
            opcion=1;
        end
    else
        write(fuente,32,410,4,"IR");
    end

    if (mouse.x>98 and mouse.x<157 and
mouse.y>407
        and mouse.y<420)
        write(fuente2,120,410,4,"MIRAR");
        if (mouse.left)
            opcion=2;
        end
    else
        write(fuente,120,410,4,"MIRAR");
    end

    if (mouse.x>194 and mouse.x<260 and
mouse.y>407
        and mouse.y<420)
        write(fuente2,220,410,4,"COGER");
        if (mouse.left)
            opcion=3;
        end
    else
        write(fuente,220,410,4,"COGER");
    end

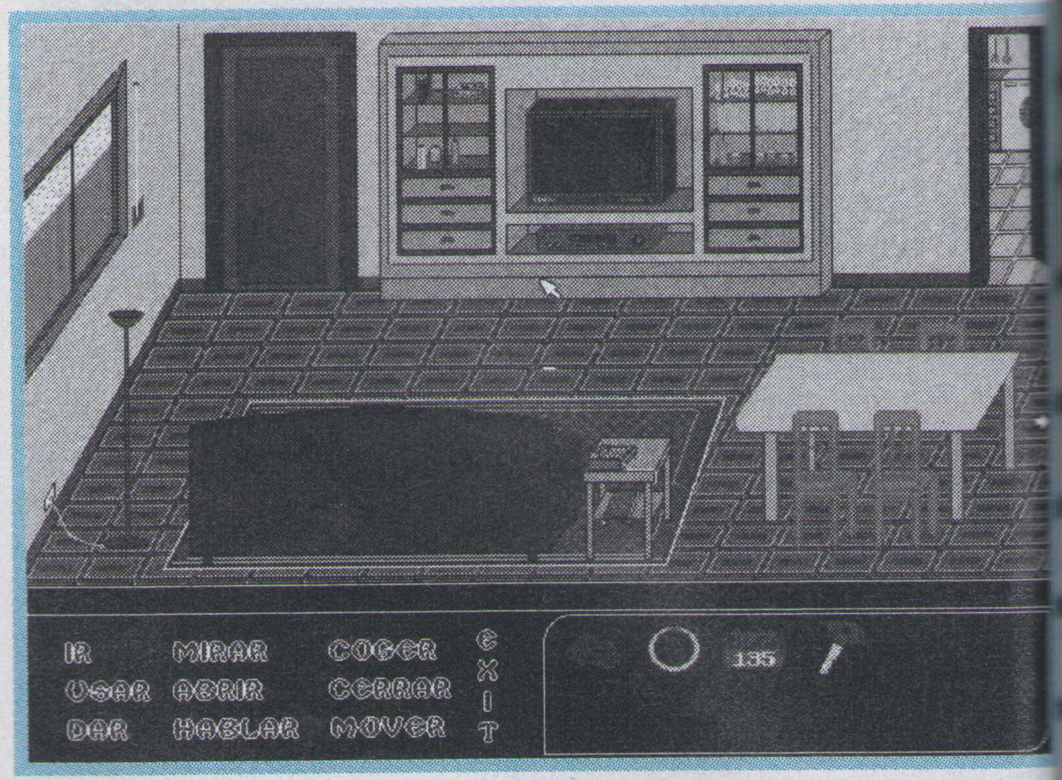
    if (mouse.x>31 and mouse.x<84 and
mouse.y>432
        and mouse.y<446)
        write(fuente2,50,435,4,"USAR");
        if (mouse.left)
            if(usar_objeto>0 or dar_objeto>0)
            usar_objeto=0;
            dar_objeto=0;
            end
            opcion=4;
        end
    else
        write(fuente,50,435,4,"USAR");
    end

    if (mouse.x>98 and mouse.x<154 and
mouse.y>432
        and mouse.y<446)
        write(fuente2,119,435,4,"ABRIR");
        if (mouse.left)
            opcion=5;
        end
    else
        write(fuente,119,435,4,"ABRIR");
    end

    if (mouse.x>194 and mouse.x<270 and
mouse.y>432
        and mouse.y<446)
        write(fuente2,225,435,4,"CERRAR");
        if (mouse.left)
            opcion=6;
        end
    else
        write(fuente,225,435,4,"CERRAR");
    end

    if (mouse.x>31 and mouse.x<70 and
mouse.y>457
        and mouse.y<470)
        write(fuente2,44,460,4,"DAR");
        if (mouse.left)
            if(usar_objeto>0 or dar_objeto>0)
            usar_objeto=0;
            dar_objeto=0;
            end
            opcion=7;
        end
    else
        write(fuente,44,460,4,"DAR");
    end

    if (mouse.x>98 and mouse.x<176 and
mouse.y>457
        and mouse.y<470)
        write(fuente2,130,460,4,"HABLAR");
        if (mouse.left)
            opcion=8;
```




```
        end
    else
        write(fuente,130,460,4,"HABLAR");
    end

    if (mouse.x>194 and mouse.x<265 and
mouse.y>457
        and mouse.y<470)
        write(fuente2,223,460,4,"MOVER");
        if (mouse.left)
            opcion=9;
        end
    else
        write(fuente,223,460,4,"MOVER");
    end

    if (mouse.x>283 and mouse.x<301 and
mouse.y>400
        and mouse.y<475)
        write(fuente2,285,404,4,"E");
        write(fuente2,285,424,4,"X");
        write(fuente2,285,444,4,"I");
        write(fuente2,285,464,4,"T");
        if (mouse.left)
            ha=0;
            clear_screen();
            hablar_con_persona=1;
            stop_scroll(0);
            pantalla=0;
            videoo=0;
            empezar=0;
            viven=2;
            graph=0;
            menu();
        end
    else
        write(fuente,285,404,4,"E");
        write(fuente,285,424,4,"X");
        write(fuente,285,444,4,"I");
        write(fuente,285,464,4,"T");
    end
end


process detecta_objetos();
private
    //se escriben aqu¡ los objetos porque no se
pueden coger
    papelera=0;
```

Autor: V. Reyes, C. López, M. López

Yamis Arena

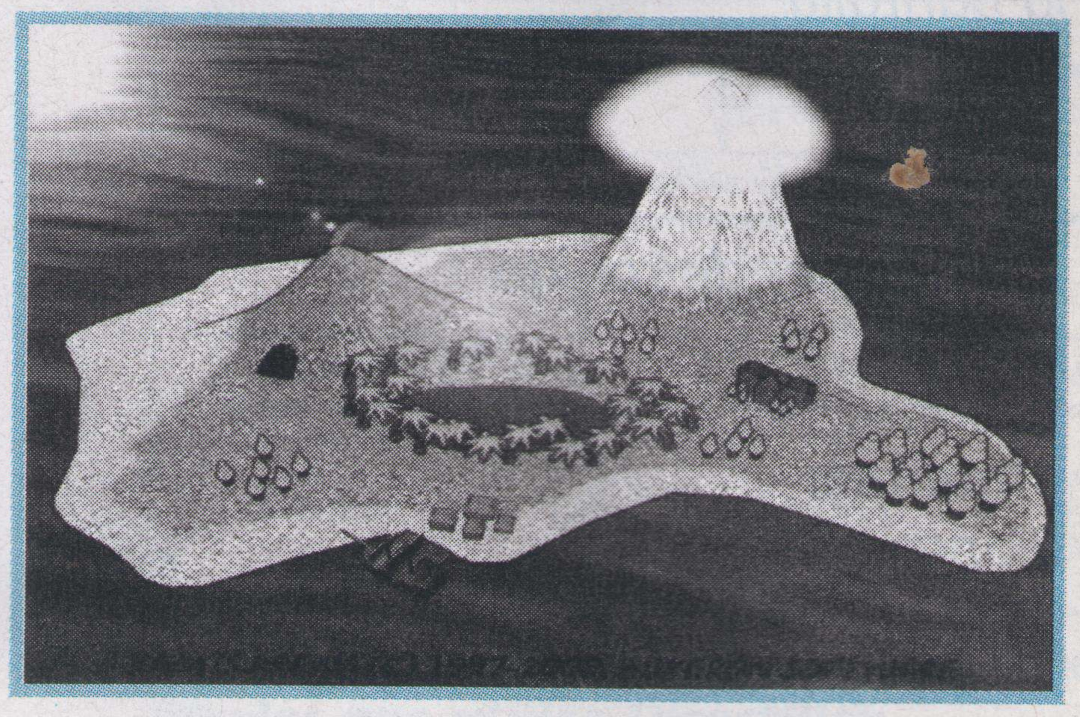
Este grupo de programadores ya se llevaron el primer premio del concurso en el número cuatro de la revista con Mr. Bones (o vete al Infierno). Ahora repiten mención honorífica con Yamis Arena un juego que se anuncia como el primero programado en Div que admite juego en red.

Nosotros en la redacción no hemos conseguido hacerle funcionar en multijugador, si lo hubiésemos hecho se hubiese llevado el primer premio, sin duda. Los autores aseguran que lo han probado en varios ordenadores y funciona, puede que sea problema de nuestros equipos. De cualquier modo, reciben el tercer premio, y recomendamos que estudiéis bien el código del juego para aprovechar su hallazgo si tenéis en mente programar algún juego con opción multijugador.

HOLA DE NUEVO

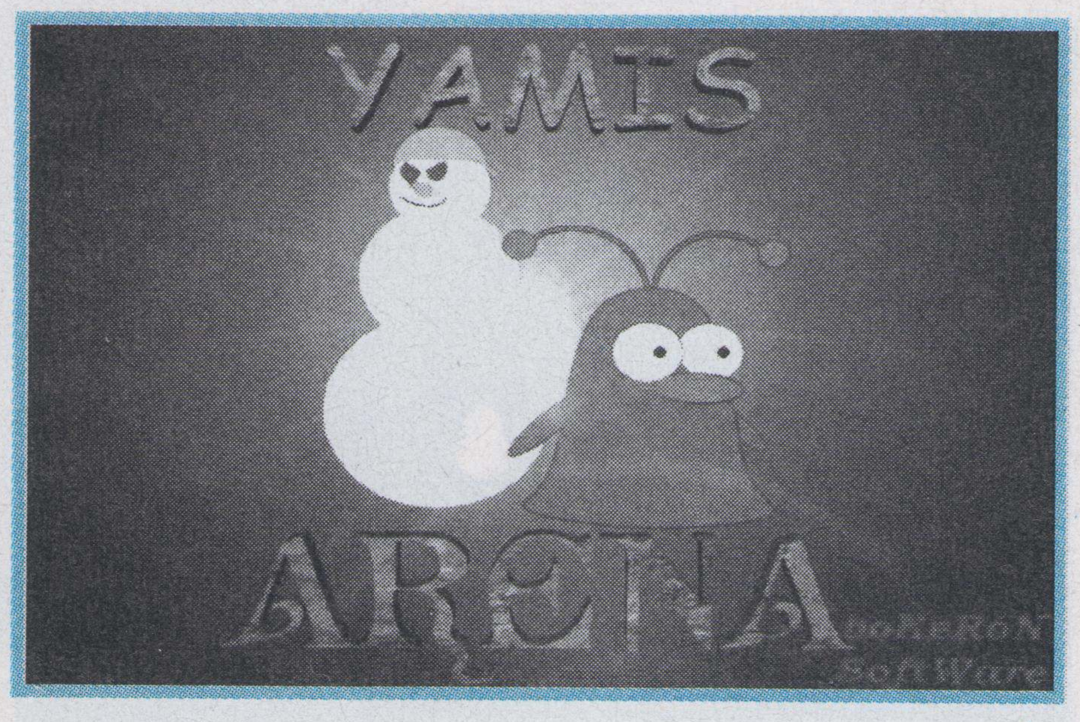
Aquí estamos de nuevo, con otro juego listo que además ¡¡funciona en red!! Ya sabemos que hay mucha gente que afirma que no funcionan, pero nosotros hemos conseguido hacerlo funcionar (de pura suerte, la verdad).

No fue nada fácil averiguar cómo se programaban juegos en red, pero finalmente lo logramos. El problema es que todo el mundo ve al servidor, pero los demás jugadores no se ven entre sí, y el servidor no los ve. La verdad probamos de todo: usar sólo el campo cero de la estructura, como el servidor, o no usarlo nunca, crear dos partidas y que cada uno fuese servidor de una, aunque esto es imposible, ya que el net.device es de sólo lectura, pero lo intentamos, queriendo hacer una especie de "pásala" con el servidor (en cada frame, el servidor pasaba a ser el otro ordenador, vamos, un caos), pero nada funcionó. Hasta que por un pequeño error de programación, más que un error un tiempo muerto en la ejecución de un programa, hallamos la solución.



Tratamos de crear dos partidas de red, cada usuario servidor de su propia partida, y que se cruzasen los datos. Para ello debían de ser servidor de una y cliente de otra. Para hacerlo, ejecutábamos el programa. Si no había ninguna partida creada, entonces se creaba una, y se esperaban quince segundos para entrar en la otra. Antes de estos quince segundos, debía de entrar el segundo jugador, que entraría como cliente del primero, creando inmediatamente otra partida en la que debía de ser el servidor (por supuesto, todo este galimatías no funcionó). Pero ocurrió algo increíble: antes de que pasasen los quince segundos para que el jugador uno se incorporara al juego ¡¡la red funcionaba!!.

La verdad es que todo esto nos resultó bastante raro, y aún ahora seguimos sin estar muy seguros de su funcionamiento, pero parece ser que hay otro error en las funciones de red, que parece ser el siguiente: cualquier estructura que tenga la misma longitud que la que se crea para jugar en red se pasa automáticamente por la red, siendo legible el campo cero de la misma para todos los jugadores, como si del servidor se tratase. Y aquí radica el secreto: crear una estructura para cada jugador (ej: net1, net2...) y que cada uno de ellos utilice el campo cero para introducir sus datos (parece ser que el primero que introduzca datos,

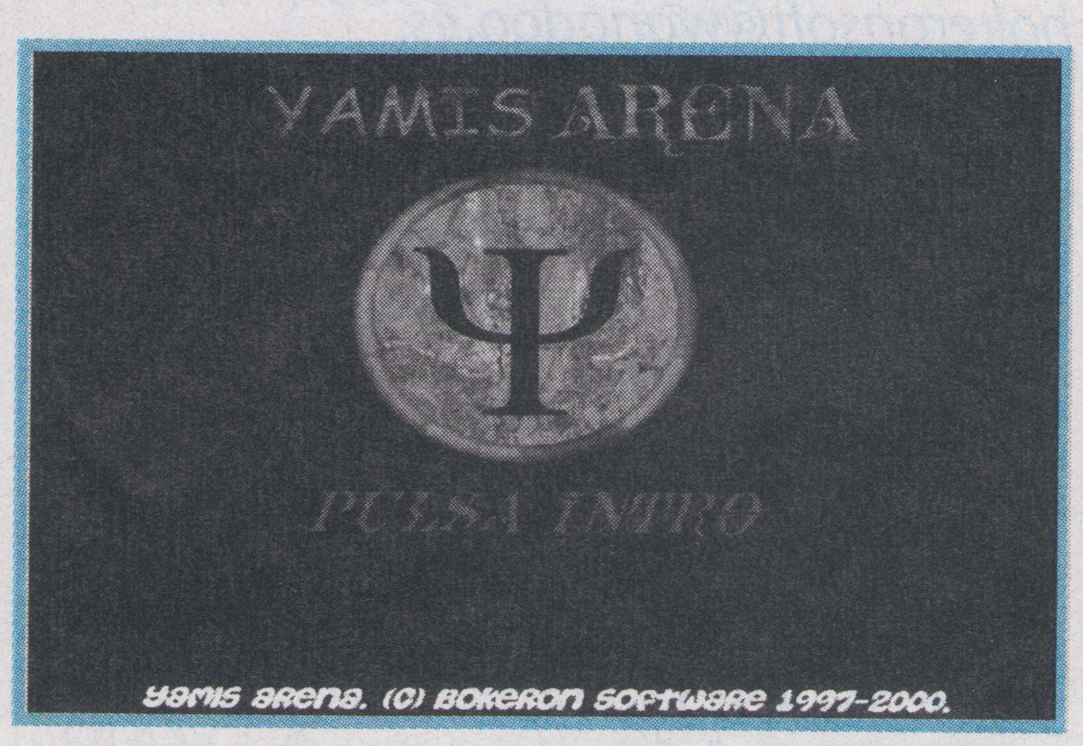


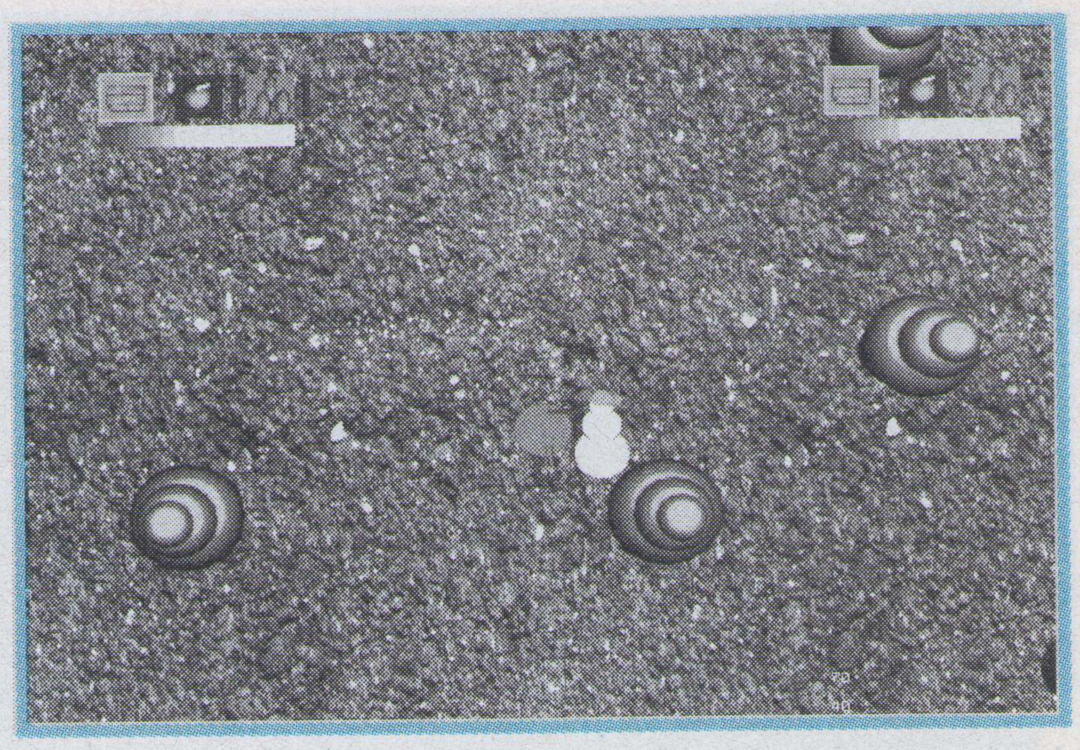
la hace suya, por lo que los demás no pueden modificarlo directamente, aunque no hemos tenido posibilidad de probar con más de dos jugadores, por lo que no podemos afirmar nada al respecto). En fin, esto es todo lo que sabemos hasta ahora. Esperamos que los cientos de Divmaníacos que hay por el mundo amplíen y corrijan nuestras observaciones.

YAMIS ARENA

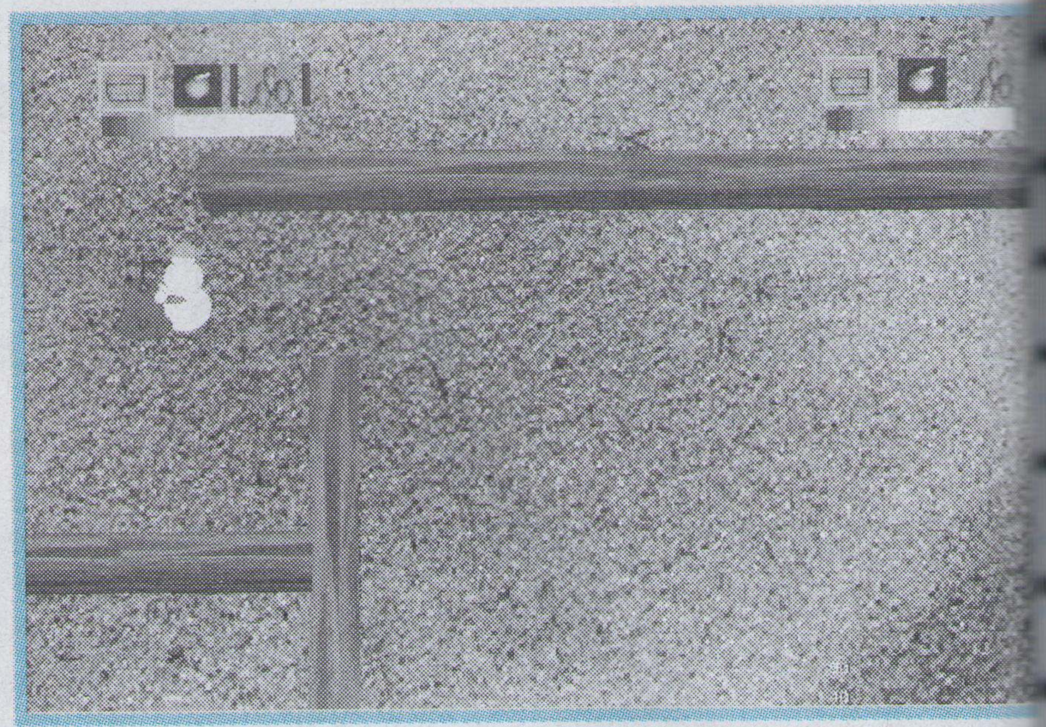
El juego es un arcade de uno contra uno, y está diseñado para dos jugadores, así que no le pidáis mucho al modo un jugador, ya que es bastante simple. La verdad, la idea de programarlo nos surgió desde que vimos el primer Div. Queríamos diseñarlo de nuevo, y ampliar las posibilidades de juego con más armas, scroll y demás. Así que, cuando salió Div 2, con posibilidad de programarlo en red, tras pensarlo detenidamente, y tras programar el *Mr Bones... O vete al infierno*, nos pusimos manos a la obra. La idea es bastante simple, pero el juego engancha bastante, sobre todo cuando dos colegas se "pican" a ver quién gana (aunque la verdad, con tan pocas armas, las estrategias y demás se agotan rápidamente, en la versión final habrá muchas más, podéis comprobar algunas de las que llevará en el *Japi quiler yamis*, la primera parte).

El juego, al ser una beta, tampoco está muy depurado, pero algunas partes, que parecen que podrían estar más depuradas, están así a propósito. Por









ejemplo, un proceso para cada jugador y para la máquina. En la versión final estamos por hacer un proceso por personaje. Sí, habéis leído bien, uno por personaje. ¿La razón?, muy simple: velocidad. Nada más con tres personajes, la cantidad de sí condicionales (IF) que incluye el proceso es enorme, tanto es así que sin ellos la extensión del proceso se-reduciría a más de la mitad. Esto influye también en la velocidad, ya que ha de hacer numerosas comparaciones y verificaciones que ralentizan la acción del juego. Cuando tenga ocho o diez personajes más el rendimiento del juego puede reducirse considerable-mente, y todo por hacer un programa más óptimo y legible para el programador. Lo importante, al menos para nosotros es el juego, y si tenemos que hacer un código más grande para que el juego sea rápido, pues lo hacemos (podrían tomar ejemplo esas grandes empresas de por ahí, que sacan algunos juegos que requieren una máquina excesivamente elevada para lo que ofrecen). Ah, esperamos que disfrutéis con la música, cien por cien original, compuesta por Carlos y Miguel. Las incluimos también como pistas de audio.

Bueno, por nuestra parte nada más. Esperamos que el tutorial para programar en red que hemos preparado os aclare todas las dudas sobre la red. Si tenéis alguna pregunta podéis escribirnos a *bokeronsoft@wanadoo.es*.

CÓDIGO FUENTE

(como siempre lo encontraréis completo en el CD-Rom)

```
PROGRAM yamis;

GLOBAL

// variables del juego para la red
STRUCT net1[1];
  vida;
  x_tio;
  y_tio;
  tio_listo;
  tio;
  endisparo;
  velocidad;
  ult_direccion;
  graf;
  selec;
  ind;
  arma;
  bala;
  bomba;
  bazooka;
  ok;
END;
STRUCT net2[1];
  vida;
  x_tio;
  y_tio;
  tio_listo;
  tio;
  endisparo;
  velocidad;
  ult_direccion;
  graf;
  arma;
  bala;
  bomba;
  bazooka;
END;
num_red;
jred;
tio1_cogido;
tio2_cogido;

// variables jugador
anim1[]=2,3,4,5,6,7,8,9,10,11,12;
anim2[]=14,15,16,17,18,19,20,21,22
,23,24;
anim3[]=26,27,28,29,30,31,32,33,34
,35,36;
ani1[]=1,2,3,4,5,6,7;
ani2[]=9,10,11,12,13,14,15;
ani3[]=16,17,18,19,20,21,22;
obstaculo;
zona;
fpg1;
arma1;
arma2;
redi_1;
redi_2;

// variables selecci¢n

op_ppal;
ind;
fpgisla;
x1_nombre;
y1_nombre;
x2_nombre;
y2_nombre;

// variables tio 1
tio1;
vel_x_1;
vel_y_1;
vida_1;
vida_total1;
vida_jug1;
cor_x;
cor_y;
x_tio;
y_tio;

// variables tio 2
tio2;
vel_x_2;
vel_y_2;
vida_2;
vida_total2;
vida_jug2;
x_tio2;
y_tio2;

// Regiones en las que vamos a dividir
la pantalla
region1;
region2;

// Variables de sonido
gong;
boom;
otraarma;
blanco;
dispara;
metralla;
```

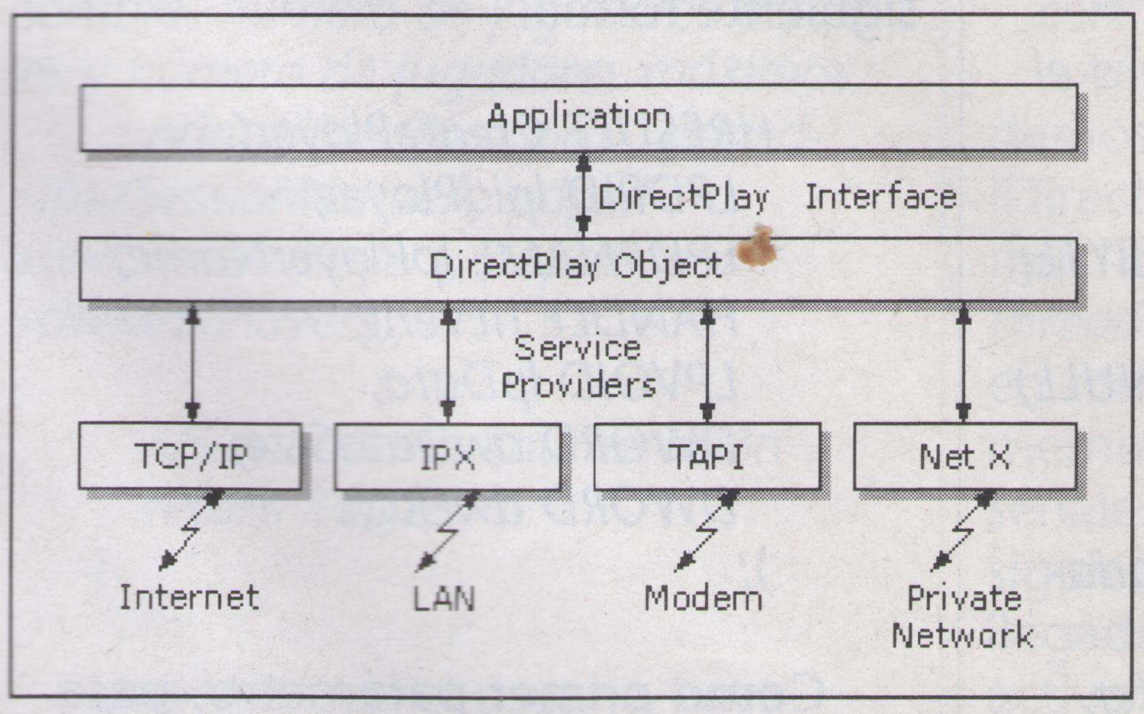
Introducción a Direct Play (1ª parte)

Múltiples posibilidades

En esta entrega analizaremos en funcionamiento general de Direct Play, el módulo de DirectX encargado de simplificar la programación de las comunicaciones y modos de juego en red.

Para aquellas personas que hayan trabajado en un sistema operativo UNIX, el concepto de *socket* seguro no les es desconocido. Se trata de abrir un zócalo en nuestra máquina, asignarle un puerto y "conectarlo" a otro abierto en otra/s máquina/s que también tiene habilitado uno en ese puerto, enviando información por él. Este sistema ha sido durante muchos años el utilizado por miles de programadores, incluso en Windows 9X y NT. Un claro ejemplo son los navegadores Internet Explorer, Netscape Communicator o el mismísimo Quake 3 Arena (en su versión Win32 como Linux).

Este es un mecanismo relativamente sencillo y eficiente para enviar o recibir datos mediante protocolos garantizados (TCP / SPX) o no garantizados (UDP / IPX), estando, incluso, orientado a tareas *multi-thread* (multihilo o multitarea) con esperas pasivas y señales o eventos, a la vez que es posible utilizarlo en varias plataformas salvaguardando la portabilidad y compatibilidad.



Aquí vemos la estructura de los diferentes niveles que utiliza DirectPlay.

Con el nacimiento de DirectX pasó algo parecido que con OpenGL. Se daba soporte a los *sockets* en Windows mediante una API llamada *WinSock*, pero se lanzaba una nueva llamada Direct Play.

Pero esta vez no era un rival en sí, sino que se apoyaba en los llamados *Service Providers*, que no son más que una serie de librerías DLL que utiliza para comunicarse con el dispositivo de red, para enviarle o recibir mensajes de él. Es decir, que esta sub API de DirectX se sitúa en un nivel bastante alto, utilizando otras APIs de nivel medio para comunicarse con la capa baja de hardware.

Su funcionamiento básico se centra en la denominada *sesión*. En alguna parte del menú del juego habrá un "Crear sesión", que indica la creación de un espacio al que se irán uniendo una serie de jugadores. Podríamos definirla como "una colección de jugadores".

Estas *sesiones* pueden ser enumeradas mediante una función de DirectPlay que veremos más tarde. Está permitido crear más de uno por máquina, con lo que es posible habilitar servidores en Internet que alberguen y gestionen más de una.

Esta API es bastante flexible en lo referente a modelos de comunicación: podemos optar entre el clásico y eficiente Cliente / Servidor, modelo distribuido punto a punto o un sistema mixto.

También es posible crear grupos de jugadores, utilizar mecanismos garantizados o no garantizados, utilizar protocolos de encriptación y certificación, soporta conexiones telefónicas, por módem y red local y *direct cable* por conexión serie o null-modem y manejo de grupos de jugadores, muticasting-broadcast, etc...

Y, por si fuera poco, los planes para DirectX 8 contemplan el envío y recepción de mensajes de voz comprimidos por Internet... Todo ello mediante un protocolo propio, el *Direct Play Protocol*.

Además, al ser una API basada en el estándar COM (Component Object Model), mediante el uso de *queries*, mantiene la compatibilidad con todas sus versiones anteriores y permite la fácil actualización a versiones posteriores.

Su única desventaja es que, de momento, a la espera de la prometedora consola de Microsoft, nVidia y otros, la X - Box, API sólo se encuentra disponible para las diferentes versiones de Windows CE / 9X / NT y 2000.

La obtención de una instancia de Directplay

Deberemos poner al principio de nuestro programa C++

```
#include <dplay.h>
#include <dplayx.h>
```

Así como incluir las librerías estáticas DPLAYX.LIB, DXGUID.LIB y OLE32.LIB

Como todos los objetos de DirectX, lo primero que hay que hacer para poder utilizarlos es obtener una *instancia* por COM mediante

```
LPDIRECTPLAY4   lpDP4 ;

CoInitialize (NULL) ;

CoCreateInstance(CLSID_
DirectPlay,
        NULL,
        CLSCTX_ALL,
        IID_IDirectPlay4A,
        (VOID**)&lpDP4);
```

La función CoCreateInstance forma parte de COM y se le pasa como primer parámetro un CLSID (*class* ID) con el número de la clase que identifica a DirectPlay. Como segundo parámetro pasamos NULL, como tercero pasamos CLSCTX_ALL, como cuarto el número que identifica a una instancia IDirectPlay4 y como quinto a un puntero donde guardarla.

Nótese que se ha hecho un *CoInitialize ()* al principio para asegurarnos de que COM se encuentra inicializado, de esta forma, *CocReateInstance* no fallará por este motivo.

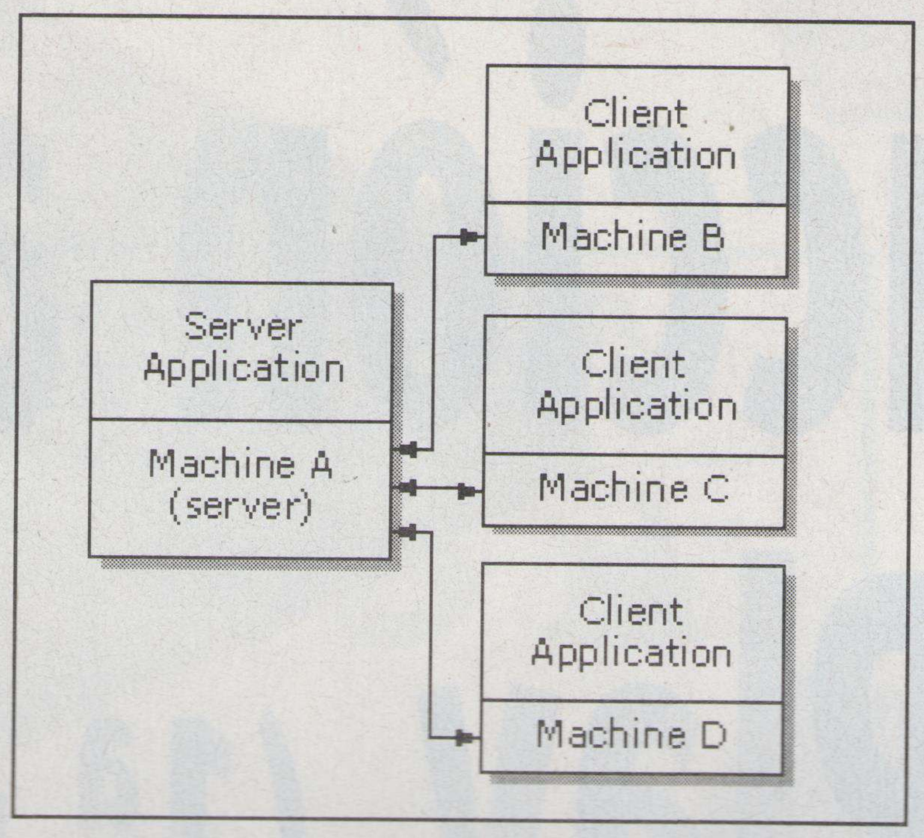
Bien, ahora que ya tenemos un IDirectPlay4, debemos enumerar los *Service Providers* que se encuentran instalados en nuestra maquina. Para ellos, hacemos

```
lpDP4 -> EnumConnections(&g_
AppGUID,
        EnumServiceProviders,
        miHWnd,
        DPCONNECTION_DIRECTPLAY);
```

Como primer parámetro, le pasamos nuestro GUID de aplicación. Si no tenemos uno, podemos generarlo con la utilidad GUIDGen que se proporciona con el paquete Microsoft Visual Studio o pasar NULL. El segundo parámetro indica la dirección de una función *callback* para recibir la enumeración, el tercero un context parameter (en este caso quiero pasarle el *handle* de una ventana List Box para que ponga el nombre del *service provider* en él), y al cuarto le paso siempre DPCONNECTION_DIRECTPLAY.

La función callback en detalle tiene la siguiente sintáxis:

```
BOOL FAR PASCAL
EnumServiceProviders (LPCGUID
pguidSP,
        VOID* pConnection,
        DWORD dwConnectionSize,
        LPCDPNAME pName,
        DWORD dwFlags,
        VOID* pvContext)
```



DirectPlay es capaz de administrar varios modelos de comunicación. Destacan el Client/Servidor y arquitectura distribuida.

```
{
    //Variables locales
    HRESULT hr;
    LPDIRECTPLAY4 lpDirectPlay4 =
NULL;
    VOID* pConnectionBuffer = NULL;
    HWND hWndListBox =
(HWND)pvContext;
    LRESULT iIndex;

    // Creamos un IDirectPlay
    CoCreateInstance
(CLSID_DirectPlay,
        NULL,
        CLSCTX_ALL,
        IID_IDirectPlay4A,
        (VOID**)&pDP4 )º;

    // Comprobamos que la cone-
xión con el Service Provider está dis-
ponible inicializándola y luego destru-
yéndola.
    if (lpDirectPlay4-
>InitializeConnection (pConnection,
0) != DP_OK) return FALSE ;

    //Como ya no necesitamos más
el lpDP4 temporal creado, lo libera-
mos (no confundir con el del principio
del artículo)
    lpDirectPlay4 -> Release () ;

    //Ponemos nombre del Service
Provider en el List Box
    iIndex = SendMessage
(hWndListBox, LB_ADDSTRING, 0,
(LPARAM)pName->lpszShortNameA);
    if(iIndex == CB_ERR) return
FALSE;

    //Reservamos memoria para el
buffer de conexión con el Service
Provider
    pConnectionBuffer = new BYTE[
dwConnectionSize ];
    if(pConnectionBuffer == NULL)
return FALSE;

    // Guardamos GUID en varia-
ble local
    memcpy (pConnectionBuffer,
pConnection, dwConnectionSize);

    //Colocamos GUID en List Box
    SendMessage(hWndListBox,
LB_SETITEMDATA, iIndex,
(LPARAM)pConnectionBuffer);

    return TRUE; // Keep enumera-
ting}
}
```

Una vez que el usuario elija el proovedor de servicios que desea utilizar (TCP/IP, modem, direct cable, etc...), inicializaremos la conexión con *InitializeConnection()* como se hizo anteriormente.

A continuación, podemos crear una sesión de la siguiente forma:

```
DPSESSIONDESC2 dpsd ;

    memset ( &dpsd, 0,
sizeof(dpsd));
    dpsd.dwSize = sizeof(dpsd);
    dpsd.dwFlags =
DPSESSION_DIRECTPLAYPROTOCOL;
    dpsd.guidApplication =
g_AppGUID;
    strcpy (dpsd.lpszSessionNameA,
"misesion") ;
    dpsd.dwMaxPlayers = 32;
    lpDP4 _ Open (&dpsd, DPO-
PEN_CREATE) ;
```

Básicamente, se rellena una estructura del tipo DPSESSIONDESC2 con el GUID de nuestra aplicación, nombre de sesión y una serie de *flags*. Obsérvese que se utilizará el *Direct Play Protocol*, lo que permite utilizar mensajes garantizados aún en el caso de estar en UDP y IPX. Esto es así porque se reenvían cada cierto tiempo. Otra ventaja de este flag es el aprovechamiento inteligente del ancho de banda o tiempo de respuesta (*throttling*).

Más abajo se especifica el máximo número de jugadores que admitirá la sesión, que serán 32, pero con un valor de 0 podemos hacer que no exista límite de participantes.

Una vez que hemos creado la sesión con IDirectPlay4: Open (), debemos añadirle un *server player*. Para ello utilizaremos el método IDirectPlay4: CreatePlayer () de la siguiente forma:

```
HRESULT CreatePlayer(
    LPDPID lpidPlayer,
    LPDPNAME lpPlayerName,
    HANDLE hEvent,
    LPVOID lpData,
    DWORD dwDataSize,
    DWORD dwFlags
);
```

Como primer parámetro, pasamos un puntero a una variable que recibirá su ID después de la

llamada. El segundo especifica un puntero a su nombre, el cuarto indica un evento que será señalado cada vez que reciba un mensaje y como sexto pasaremos la constante DPPLAYER_SERVERPLAYER, retornando DP_OK si todo fue correcto.

Bien, si estuviésemos en un modelo Cliente / Servidor, habríamos creado ya éste último. Nos faltaría crear un *player* cliente uniéndolo a la sesión. Conviene enumerar todas las sesiones existentes y preguntar al usuario/a cuál quiere unirse. Para ello, contamos con el método IDirectPlay4: EnumSessions (). Su uso puede verse en el siguiente ejemplo:

```
//Enumera todas las sesiones
LPDPSESSIONDESC2  dpsd ;

memset (&dpsd, 0, sizeof(dpsd));
dpsd.dwSize = sizeof(dpsd);
dpsd.guidApplication =
g_AppGUID;

lpDP -> EnumSessions (&dpsd,
10000, EnumeraSesiones, NULL,
DPENUMSESSIONS_ALL);
```

Como vemos, después de rellenar una estructura DPSESSIONDESC2 con el GUID de nuestro programa y su tamaño, se la pasamos a dicho método, junto con el *timeout* (10 segundos aquí), la dirección de memoria de una función *callback* y un *flag* que le indica que enumere todas las sesiones, admitan o no más jugadores.

Aquí podemos ver en detalle el *callback*:

```
BOOL FAR PASCAL
EnumeraSesiones (LPCDPSESSION-
DESC2 pdpsd, DWORD* pdwTimeout,
DWORD dwFlags, VOID* pvContext)
{
  // Si pasó el timeout, para la
enumeración
  if (dwFlags & DPESC_TIME-
DOUT) return FALSE;

  // Se ha encontrado una sesión
correcta. Imprime por pantalla su
nombre, número de jugadores actua-
les y número de jugadores máximos.
  printf("%s (%d/%d)", pdpsd-
>lpszSessionNameA, pdpsd-
>dwCurrentPlayers, pdpsd-
>dwMaxPlayers);

  //Continúa la enumeración
  return TRUE;
}
```

Una vez que el usuario haya escogido en qué sesión quiere entrar, lo unimos a ella con:

```
LPDPSESSIONDESC2 dpsd ;

memset (&dpsd, 0, sizeof(dpsd));
dpsd.dwSize = sizeof(dpsd);
dpsd.guidApplication =
g_AppGUID;
dpsd.guidInstance = pSession-
>guidSession;
lpDP4 _ Open (&dpsd, DPO-
PEN_JOIN) ;
```

Nótese que hemos de indicar el GUID de la sesión a la que queremos unirnos.

El paso siguiente será dar al nuevo jugador de alta. De nuevo, volveremos a utilizar el método IDirectDraw4: CreatePlayer (), tal y como lo hicimos anteriormente.

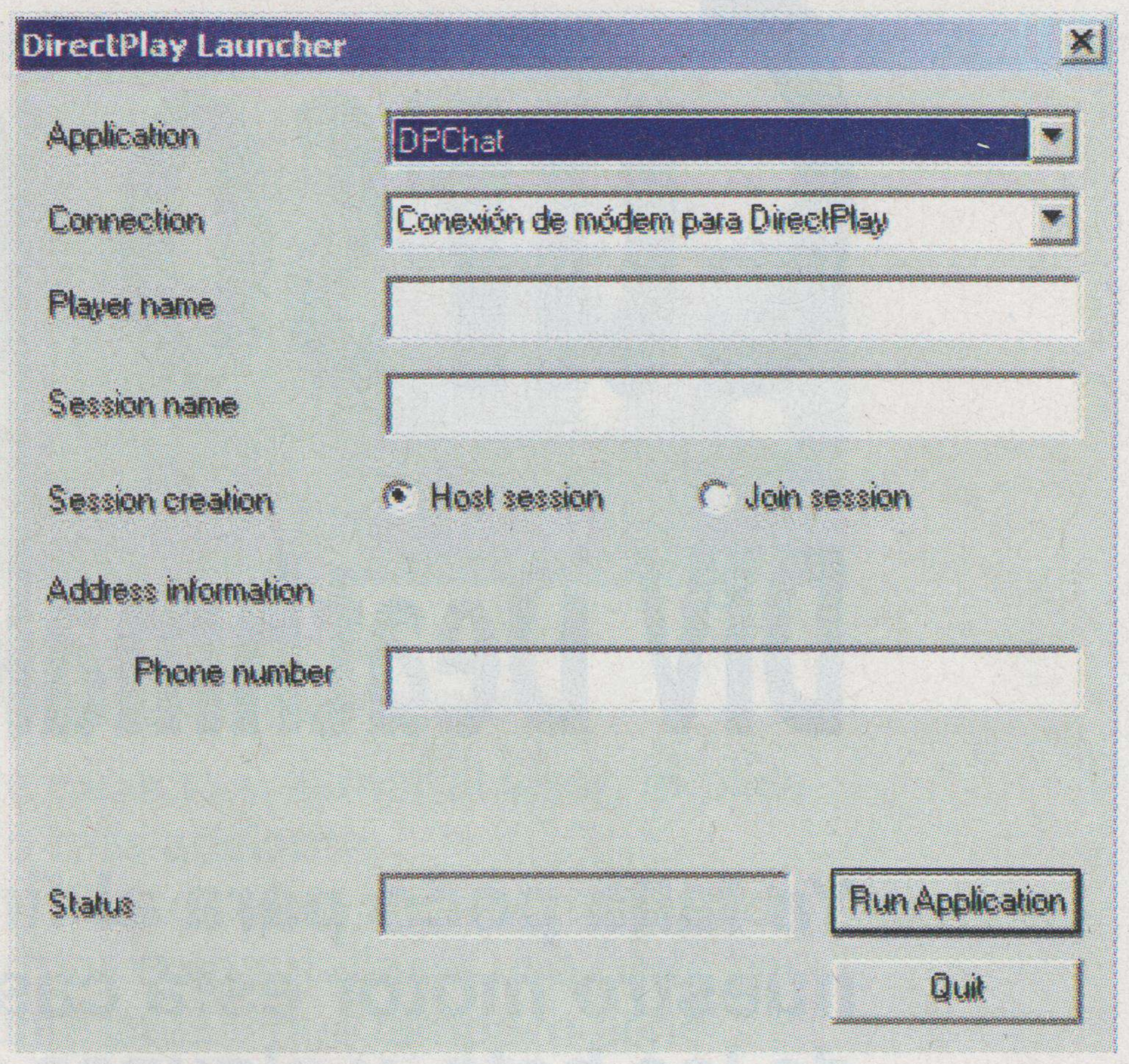
Ahora ya podemos enviar mensajes de uno a otro mediante IDirectPlay4: Send ()

```
HRESULT Send(
  DPID idFrom,
  DPID idTo,
  DWORD dwFlags,
  LPVOID lpData,
  DWORD dwDataSize
);
```

Los dos primeros parámetros especifican el ID del remitente y destinatario. Como *flags* podemos especificar:

- DPSEND_GUARANTEED: Si queremos garantizar la llegada del mismo. No importa que estemos utilizando un protocolo UPD o IPX. Como activamos el DPSESSION_DIRECTPLAYPROTOCOL al crear la sesión, aún tratándose de un sistema no garantizado, se asegurará la recepción y envío de mensajes.
- DPSEND_ENCRYPTED: Podemos cifrar los mensajes, para lo cual, al crear la sesión, debimos especificar un *flag* indicando que se trata de una *secure session*.
- DPSEND_SIGNED: Por si no fuera suficiente con una sesión segura, DirectPlay nos brinda la posibilidad de firmar digitalmente los envíos, para lo que tendremos que tener instalado en nuestra máquina algún mecanismo de certificado.

A la hora de recibir mensajes se desaconseja el uso en bucle de IDirectPlay4: Receive (), siendo mucho más eficiente un sistema de *threads* pasivos, debiendo utilizar en muchos casos mecanismos de memoria compartida, semáforos y señales. El funcionamiento es el siguiente: un hilo despierta por la llegada de una señal externa que activa un evento. Es ese momento podemos leer el mensaje entrante de la cola con




```
HRESULT Receive(
  LPDPID lpidFrom,
  LPDPID lpidTo,
  DWORD dwFlags,
  LPVOID lpData,
  LPDWORD lpdwDataSize
);
```

Los dos primeros parámetros indicarán el remitente y destinatario. Como *flags* podemos especificar:

- DPRECEIVE_ALL - Coge el primer mensaje de la cola y lo borra de la misma.
- DPRECEIVE_PEEK - Coge el primer mensaje de la cola sin borrarlo de la misma.
- DPRECEIVE_TOPLAYER - Activa destinatario. Sólo se cogerán los mensajes dirigidos a él, ignorando los demás.
- DPRECEIVE_FROMPLAYER - Activa remitente. Sólo se cogerán los mensajes enviados por el remitente seleccionado, y se ignoran los demás.

Conclusiones

Esto no ha hecho más que empezar. Con DirectPlay podemos hacer infinidad de aplicaciones: desde el más simple programa de *chat* hasta el sistema de red del juego más complicado, pero debido a la intención meramente introductoria de este artículo, no hemos querido profundizar más. En próximas entregas veremos más detalladamente su funcionamiento, añadiremos como ejemplo un pequeño programa y analizaremos algunas características avanzadas, tales como el manejo de grupos o la creación de *lobbies* para servidores en Internet.

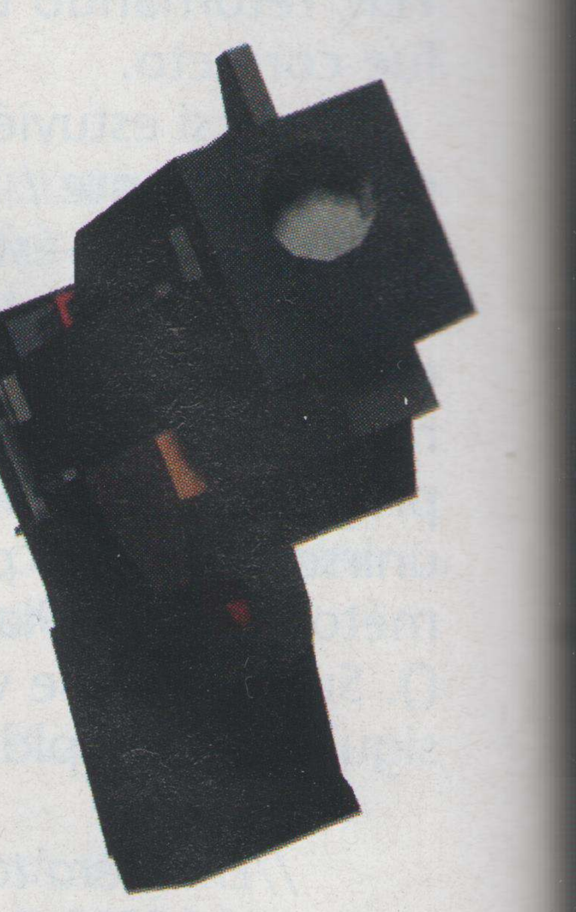
De todas formas, si tenéis alguna duda, queréis hacer algún comentario o simplemente intercambiar ideas, podéis encontrarme en *santyhammer@latinmail.com.*

Santiago Orgaz

Curso de juegos en 3D

DIV Deathmaker (V)

Ya falta poco para el final. En este número, nuestro motor está casi al completo. Salvo la IA de los bots, y algunas partes como son los disparos/explosiones y los daños en el jugador y bots, todo lo demás está listo.



Tras haber acabado completamente con la "interfaz" de los menús y, prácticamente todo, salvo el motor del juego (la parte 3D), podemos meternos en profundidad en lo que es la programación pura y dura de juegos. Hasta ahora no había una clara diferencia entre la línea que separa la programación orientada a juegos y la de los programas, pero como notaréis, en esta última la programación toma una orientación distinta.

Lo primero que se puede observar en nuestro nuevo PRG (el cual ha crecido de forma espectacular) es que muchas partes han sufrido cambios, en especial, la declaración de variables. Como todos sabréis, al empezar un programa se tiene una base mental sobre la estructura general que va a tener, pero a medida que se avanza en el programa esta base que se fijó va a ir modificando y ampliando. Con DDM no ha podido ser de otro modo; son muchas las modificaciones como para enumerarlas todas, pero por ejemplo:

Algunos datos se guardaban en variables tipo PRIVATE. Al comprobar más tarde que era necesario acceder a esas variables desde otro proceso, se han tenido que cambiar algunas a GLOBAL, o LOCAL... también pasa lo mismo con variables que se guardaban en tablas tipo WORD que más adelante, al observar que era necesario acceder a su OFFSET (desde un write_int, por ejemplo) se han tenido que pasar a variables "sueltas".

El portal se hace más grande

Nuestra función *portalmuerte()* ha aumentado (y bastante) en código. Antes de nada, quiero comentar que HABÍA UN ERROR en el PRG de la anterior Divmanía ("mea culpa", lo siento), exactamente en la asignación de registros de PLAYER[7], parte que consideré como complicada y quiso la fortuna que justo ahí se diera un BUG. En este número ya está corregido.

En el anterior número la función *portalmuerte()* cargaba y preparaba perfectamente el modo8 y, de hecho lo sigue haciendo, aunque se pueden apreciar los cambios a partir del fin del bucle principal de juego: ahí comienza un largo proceso de DESCARGA de todo lo que había en memoria y no era necesario. De todos modos, hay un problema de memoria descrito más adelante.

Otra novedad en *portalmuerte()* es una parte que gestiona unas variables llamadas "mensaje1", "mensaje2", etc... explicado en la sección "EL MSJMASTER".

Por último, os quiero recordar que *portalmuerte()* TODAVÍA no guarda los resultados de una partida, por razones obvias. Todavía no se puede "matar" a nadie, porque las armas no disparan perfectamente (aunque sí hacen la animación y llaman al proceso disparo) y no está implementado el reconocimiento de colisiones y daño.

El MSJMASTER

Este nombre tan curioso es el que alberga uno de nuestros procesos, además uno muy importante, se trata del "Maestro de los mensajes", es decir, el proceso que gestiona la "cola" (o pila) de mensajes, añadiendo mensajes nuevos, reteniéndolos por un tiempo y eliminando los antiguos cuando se sobrepasa el límite.

En realidad, el *msjmaster()* funciona en conjunto con *portalmuerte()*, puesto que es esta última función la que se encarga de ir borrando mensajes con el tiempo y mostrarlos a cada FRAME. *Msjmaster()* simplemente se encarga de los mensajes nuevos.

Cuando queremos meter un mensaje en la cola llamamos a *msjmaster*, por ejemplo, con la siguiente sintaxis: *msjmaster("Este es un mensaje de prueba")* y *msjmaster* comprueba si hay un "slot" o mensaje libre en orden y lo introduce allí; si no encuentra ningún mensaje libre (o mejor dicho, línea de mensaje), corre todos un lugar hacia arriba y lo coloca el primero.

Como podéis observar, ya están implementados todos los mensajes de recogida de objetos y armas. Faltan los mensajes que indican las muertes y asesinatos del jugador y los bots.

Contrólame a ese jugador

Llegamos al proceso más "escabroso", por decirlo de algún modo, del programa. Algunos pueden pensar que en la parte de la IA de los bots es necesario extenderse más que en el proceso del jugador. En cierto modo así es, sin embargo, no hay tanta diferencia, porque lo que hacen los bots está de cierta manera "escondido"; no necesita representación en pantalla, mientras que lo que hace el jugador sí. Por supuesto, los bots también tienen que estar un tiempo cambiando de arma y recargando (aunque en algunos juegos parece ser que cambian instantáneamente de arma).

El proceso del jugador es muy largo como para explicarlo en su totalidad. Hay varias partes que siguen una estructura similar, así que nos basaremos en la composición general del proceso. Por supuesto hay que recordar que todavía está incompleto, aunque no falta mucho.

Empezamos con un esquema básico del proceso jugador actual que podéis ver en la Tabla 1.

Como veis, son varias partes,

todas y cada una de ellas imprescindibles para el correcto funcionamiento y movimiento del personaje por el mundo 3D. Ahora, de forma más detallada, os explicaremos cada una de ellas:

La preparación del proceso: consiste básicamente en la declaración de las variables privadas y la asignación de los valores iniciales del jugador para el modo8. Por ejemplo, las constantes de m8.height, m8_step, radius, etc... en cuanto a los valores iniciales, podemos encontrar "como=1", variable que en ese valor nos indica que el jugador está "vivo"; también el teletransporte a una flag de inicio al azar (con su correspondiente efecto de teletransporte, basado en procesos "dot") y la inicialización de variables como la vida (a 100), la munición, el arma escogida por defecto...Os aconsejo que le echéis un vistazo a los comentarios que hay en la declaración de variables privadas de jugador(), ya que aclaran bastante su utilidad.

Muestra de la munición, vida y blindaje: son unos simples writes, hay que considerar que el que muestra la munición tiene que hacer un write_int (y necesita un dato que no sea tabla, de ahí la existencia de mun_vab), y tiene que conocer qué arma hay seleccionada y si no se está cambiando de arma.

Restauración de pantalla: seis líneas banales que no permiten que se quede la pantalla en el fade cuando se cogen objetos, armas, etc.

Gestión del Quake Damage: perfectamente explicado en el listado; esta parte controla la variable COUNTER (contador de Quake Damage) y se encarga de terminar el Quake Damage al alcanzar el límite, para ello muestra un mensaje.

Gestión de la gravedad: una parte algo larga y quizá complicada. Toma las alturas del techo y el suelo y las divide en dos partes. Una de ellas, si se toca suelo (la variable Z es igual a la altura del suelo) o no, y la otra, si se toca techo. En la primera se controla el "balanceo de visión" (que se produce al saltar desde muy alto; la vista se va desplazando hacia abajo) y se resetea la gravedad a 0 o, en caso de que no se toque suelo, se aumenta el balanceo y la gravedad. En la segunda, si se toca techo (obviamente sólo se puede hacer tras un salto) la gravedad se restablece a 1. ¿Por qué? Por si todavía quedara gravedad negativa (vamos, que sube el proceso), para que no se quede "pegado" al techo un rato, sino que nada más tocar el techo comience a bajar.

Control del salto: es muy simple. Si se pulsa el botón derecho del ratón y se está tocando el suelo (porque si no podríamos volar), la gravedad adopta un valor negativo (-25) y el balanceo de visión se restablece (100).

Control de pulsación de teclas: aquí hay que explicar cómo funciona "armaa". Armaa es una variable que indica el arma seleccionada y su estado. Así, el valor 10 corresponde al láser de impulsos en estado "normal", el 11 a "sacando el láser de impulsos", 12 sería "recargando el láser de impulsos", 13 "esperando tras disparo de láser de impulsos", 14 "disparando el láser de impulsos". Con la Qk22 sería con la decena del 20, la ADC la del 30, etc. No todas las armas usan todos estos valores, de todos modos.

En esta parte, lo que se hace es comprobar si se pulsa una tecla de las de selección de armas (1-7) Y si "armaa" termina en 0 o en 1 (arma lista o cambiando de arma). Para hacer esta comprobación se usa el MOD (resto). IF ((Armaa MOD 10)<2) significa eso mismo, si el resto de la división "armaa entre 10" es menor que 2... Si se cumple la condición, se puede cambiar de arma, y por ejemplo si se pulsa 3 (ADC) "armaa" pasa a valer 31. Otra parte del proceso se encargará de guardar el arma actual y sacar la ADC.

Control de dirección: es algo largo pero básico y fácil de entender. El proceso del jugador avanza, luego comprueba si ha colisionado con algún bot o alguna caja explosiva mediante los get_id y si es así, retrocede lo avanzado. Funciona igual para el avance hacia delante, atrás, y el "strafe".

Control de colisiones con objetos y armas: es realmente la parte más larga pero también es muy fácil de entender. Si se está tocando suelo (único modo del que se puede recoger un objeto/arma) se van comprobando las colisiones con los distintos objetos y armas, uno por uno, y si se choca con alguno se "coge" (añadiendo el arma, la munición o lo que sea), se hace un fade y se muestra el mensaje de "recogida", diciéndole al proceso del objeto o arma que ha sido recogido, para que "desaparezca" temporalmente.

Actualización del aiming: son sólo tres líneas que registran el movimiento del ratón para mover el centro de visión. En la última línea, se guarda el ángulo con el que se apunta en "oldangle" (para cuando el proceso estuviera "volando", ya que tiene dos ángulos, uno el ángulo de dirección en que se avanza y otro el ángulo en el que se mira).

Control del visor: cuatro escasas líneas para detectar la pulsación de espacio y llamar al visor, explicado más adelante.



Visión espléndida del magnánimo búnker.

Gestor de la representación de armas en pantalla: es una parte bastante larga y que tampoco tiene tanto misterio, pero sería extenderse demasiado. Os aconsejo que lo miréis en el código fuente, ya que no es tan difícil. De todos modos he considerado esta parte como un "adorno" o "retoque profesional" al juego, ya que el modo de cambiar las armas, tan suave, y el balanceo de las mismas, no son estrictamente necesarios para que el juego funcione).

Control de disparo: está mezclado con la parte anterior porque es algo muy parecido, sólo que además de representar el arma llama a la función disparo con los parámetros pertinentes, indicando primero el tipo de disparo (0=láser de impulsos, 1=qk22/ADC, 2=bláster de fusión, 3=repetidor láser, 4=lanzamisiles, 5=mina).

Comprobación del máximo de municiones: a veces se supera el máximo de municiones, por ejemplo si se tienen 149 balas y se coge un cargador, no deberían aumentar a 159 sino a 150; en realidad aumentan a 159 pero esta parte lo corrige después, reduciéndolas a 150.

Control de la muerte: es una parte que, aunque funciona, no se puede usar todavía, así que lo dejaré para el próximo número) paciencia.

Restauración del ratón y caída en gravedad: el ratón se vuelve a centrar en pantalla y se cae (o se sube) la gravedad.

Con esto acaba la visión global del proceso que controla al jugador. ¿A qué no era tan complicado como parecía?

El visor

¿Qué es el visor? Os hago una pequeña introducción... en Div Deathmaker se puede escoger un skin para cada bot, pero si sólo se tiene un skin o poca memoria, es casi seguro que exista más de un bot con el mismo skin. ¡Vaya! ¿Cómo diferenciarlos al jugar con dos iguales? Con el visor. Éste es un proceso que manda una "sonda" en la dirección en que está mirando el jugador y al acercarse a un bot retorna su



Lanzamisiles y, enfrente, Quad Damage. Como podéis comprobar los mensajes no se cortan un pelo.

nombre. Es un proceso lento, por lo que sólo se activa al pulsar la tecla ESPACIO. Y es el único modo de diferenciar bots con el mismo skin, además de por su comportamiento.

En el PRG ya se avisa que visor es un proceso inestable debido a los propios bugs del modo8 y no se aconseja usarlo en exceso... por desgracia no puedo hacer nada para solucionarlo.

Los objetos y las armas

Los procesos que controlan los objetos y las armas son procesos muy simples y fácilmente entendibles. Simplemente, al ser llamados por *portalmuerte()*, se teletransportan al número de bandera que se le introduce como parámetro. Si no existe, desaparecen inmediatamente (porque no se definió esa bandera). En caso de existir la bandera "entran" en el modo8 y permanecen ahí. Aprovechan su variable local ANGLE para saber cuando han sido cogidas y cuando no. ANGLE es un contador que va aumentando a cada FRAME. Cuando un bot o el jugador coge el objeto (las armas no, porque siempre se pueden recoger si no se tienen ya) su ANGLE se reinicia, permaneciendo "escondidas" hasta que ANGLE alcanza su valor de contador y reaparecen, con un efecto de "reaparición" mediante procesos dot.

Los procesos DOT

Ya han aparecido mencionados antes estos procesos. ¿De qué se trata? Aquéllos que sepan qué significa dot pueden haberlo adivinado. Dot, en inglés, significa "punto". Son procesos que muestran un gráfico de UN PÍXEL, moviéndose en una dirección aleatoria en el mapa 3D. Sirven para efectos de teletransporte, choque de bala, impacto, sangre, explosión, etc... si habéis jugado a algún juego como Quake, que usa este efecto de "lanzar pixeles" usándolos como sprites pequeños, entenderéis mejor la utilidad de los procesos dot. Si no, podéis probar a coger un objeto (que no sea el Quad Damage, que tarda el doble en reaparecer) y quedaros mirando en dirección a donde estaba. Cuando aparezca veréis algunos destellos aparecer... son procesos dot.

Tabla 1: Estructura del proceso "jugador()"

Preparación del proceso, teletransporte de inicio.

COMIENZO DEL BUCLE:
Muestra en pantalla de munición, vida, blindaje.
Restauración de la pantalla (si había FADE).
Gestión del QUAD DAMAGE.
Gestión de la gravedad (parte importante).
Control del salto.
Control de pulsación de teclas 1-7 (para las armas).
Control de dirección (importante también).
Comprobación de colisiones con objetos y armas.
Actualización del "aiming" (dirección en que se apunta).
Control del "visor" (comprobador de bots, explicado en sección "El visor").
Gestor de la representación de armas en pantalla:

- Punto 1: muestra del cambio de arma.
- Punto 2: muestra del "balanceo" (efecto de movimiento).
- Punto 3 –externo a esta parte-: control del disparo (parte importante).
  Intercambio de armas al acabarse la munición en una.
- Punto 4: muestra/control de la Qk22 cambiando de cargador.
- Punto 5: muestra del arma disparando.

Comprobación de que las municiones no superan los máximos.
Control de la muerte del jugador (implementado pero no utilizable todavía).
Restauración del ratón y caída en gravedad.
FIN DEL BUCLE Y DEL PROCESO

Apurando la memoria

Div Deathmaker no es un programa con requerimientos muy elevados. De hecho, el ordenador en que yo mismo lo programo es un P150 con 16 Mb de RAM, y eso tiene su parte buena: no puedo hacer un juego que no me funcione a mí mismo. Como algunos sabréis y habréis vivido ya, los programadores con recursos altos (Pc's muy potentes) tienden a desperdiciar memoria (en parte por "vaguería", en parte porque al no leer "se ha quedado sin memoria", no saben que están desaprovechándola)... Div Deathmaker funciona perfectamente con 16 Mb de RAM. Pero sólo una vez. Esto es confuso, pero tiene su explicación:

La memoria anda justita en el juego porque requiere gran cantidad de gráficos y Div consume muchos recursos. Pero hay un problema, la primera vez que se inicia una partida todo va de maravilla, pero una vez se termina y se quiere iniciar otra da un problema de falta de memoria (sólo con 16 Mb de RAM)... debido a LOAD_WLD. Esta función no dispone de una UNLOAD_WLD, y no es posible descargar un mapa 3D cargado con anterioridad, con lo que ahí radica el principal problema de pérdida de memoria en los juegos con modo8. Es un obstáculo irremediable y al que nos tendremos que enfrentar siempre.

¡Queremos jugar ya!

Esa es una de las frases con las que me encuentro a menudo. He intentado acelerar bastante esta parte, porque no es muy complicada, y dejar para el próximo número algo tan "relativamente" simple como los disparos y las explosiones (y sus respectivas interacciones con el jugador y los bots), el "tablón de puntuaciones" (accesible con una rápida pulsación de TAB) y la intrigante y querida IA de los bots, que tanto estáis esperando. No puedo asegurar que lleguemos a ver toda la IA en un solo número, pero procuraré que todo lo que esté en el próximo número sea ya jugable, aunque con carencias.

Y por supuesto, no me he olvidado de los sonidos y la música. Pero eso, como es lo habitual, lo dejamos para lo último.

Recordad que si tenéis alguna sugerencia/duda/o lo que sea me lo podéis mandar a mi e-mail las 24 horas del día los 365 días del año. ¡Hasta el próximo número!

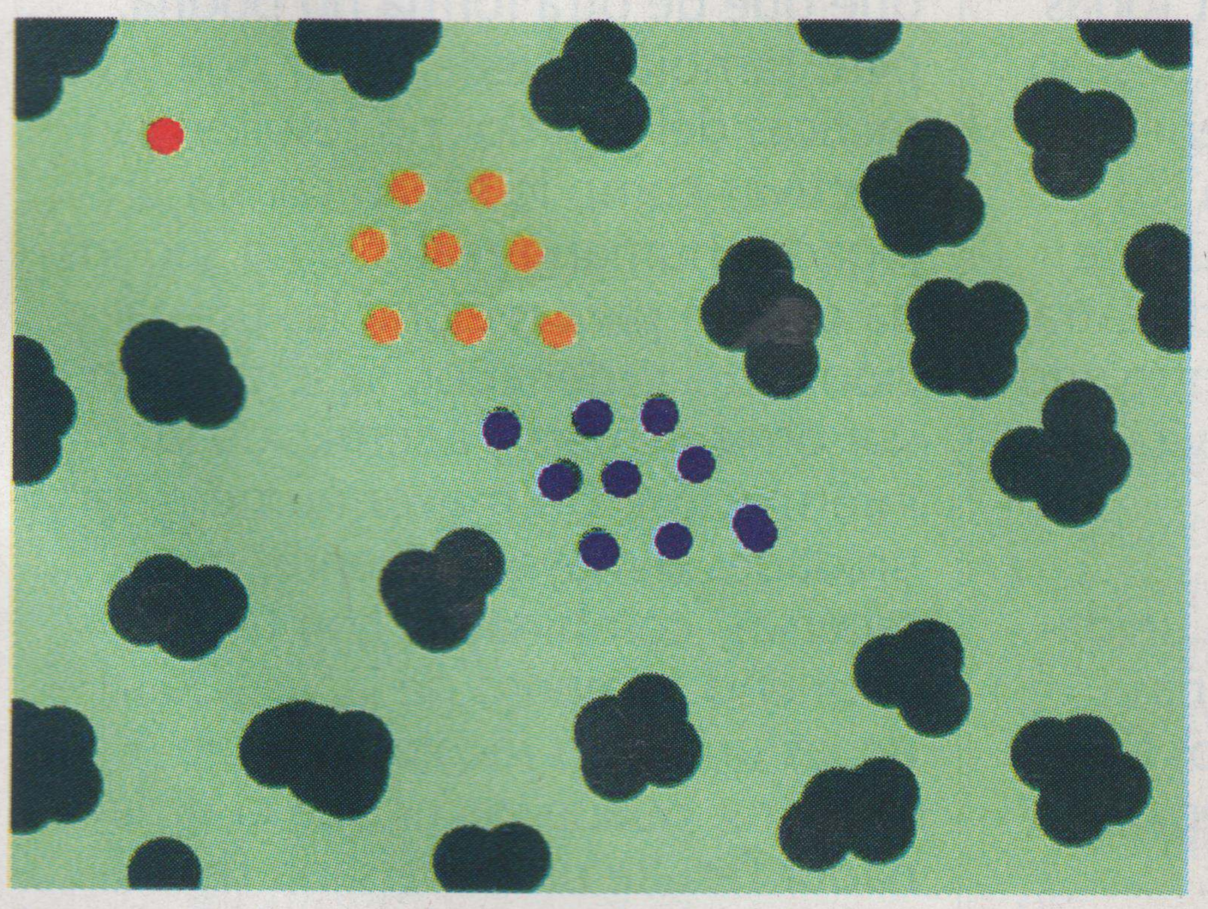
Ferminho (Fermín Vicente)
ferminho@lycosmail.com

Programación de Juegos de Estrategia – 9

Principios básicos de IA (II)

Como decíamos el mes pasado, ahora iremos desarrollando las bases del sistema controlador; éste se encarga de manejar las unidades como si fuese un jugador humano enfrente del ordenador.

Es decir, es el que le dice a las unidades los objetivos que deben atacar, cuándo y cómo hacerlo. También es el que construye edificios en función de las necesidades de recursos. Una de las cosas que debe hacer el sistema controlador es construir los edificios que vaya necesitando. Esto significa que debe tener una política de prioridades. Si deseamos un adversario principalmente defensivo, concentrará sus esfuerzos en reforzar su terreno y cuando lo haya hecho, será cuando empiece a organizar una ofensiva. Si, por el contrario, deseamos que sea un adversario ofensivo, deberá construir muchas barracas para poder crear muchas unidades en poco tiempo. Tan sólo dedicará un pequeño porcentaje del tiempo y de los recursos a reforzarse.



Los naranjas están en inferioridad porque una de sus unidades (la roja) está demasiado lejos.

Control de las acciones realizadas

Para poder hacer que el ordenador reparta sus recursos en defensa o ataque, debemos conocer ciertos parámetros. En primer lugar, hay que saber los parámetros correspondientes a los recursos. Deberemos fijar una medida de tiempo para poder calcular los porcentajes que dedicados a cada actividad. A partir de ahora, llamaremos UT a la unidad de tiempo (1 UT puede ser por ejemplo equivalente a 10 FRAMES). Debemos tener una tabla en la que se guarden las UT's utilizadas en cada actividad, por ejemplo, construyendo o investigando. También nos sirve para calcular los ingresos de algún recurso por cada UT, porque saber que tenemos X unidades de oro si no sabemos cuánto estamos ganando cada cierto tiempo, no nos indica gran cosa. Si conocemos cuánto oro obtenemos por cada UT, podremos limitar la producción de unidades en función de su coste, o si lo que queremos es construir más unidades, podemos saber cuántas UT's necesitaremos o, podremos crear más fuentes de ingresos si los consideramos necesarios.

Otra utilidad para medir el tiempo es la posibilidad de hacer que ciertos eventos ocurran al rato de empezar a jugar o con cierta periodicidad durante el juego. Resumiendo, el ordenador únicamente puede procesar información, por lo que cuánta más le proporcionemos para procesar, más "inteligente" será nuestro controlador.

Cómo manipular la información obtenida

De momento, conocemos la posición en el espacio, la posición en el tiempo y el estado (andando, huyendo, espiando, etc.) en el que se encuentra cada unidad, así como la posición de cada una (tanto amigas como enemigas) y su cercanía (dentro de su campo de visión). Además, conocemos ciertos datos sobre cada unidad, como puede ser la vida que les queda. Con estos datos, podemos calcular el poder de una unidad, gracias a lo cual podremos comparar dos unidades entre sí.

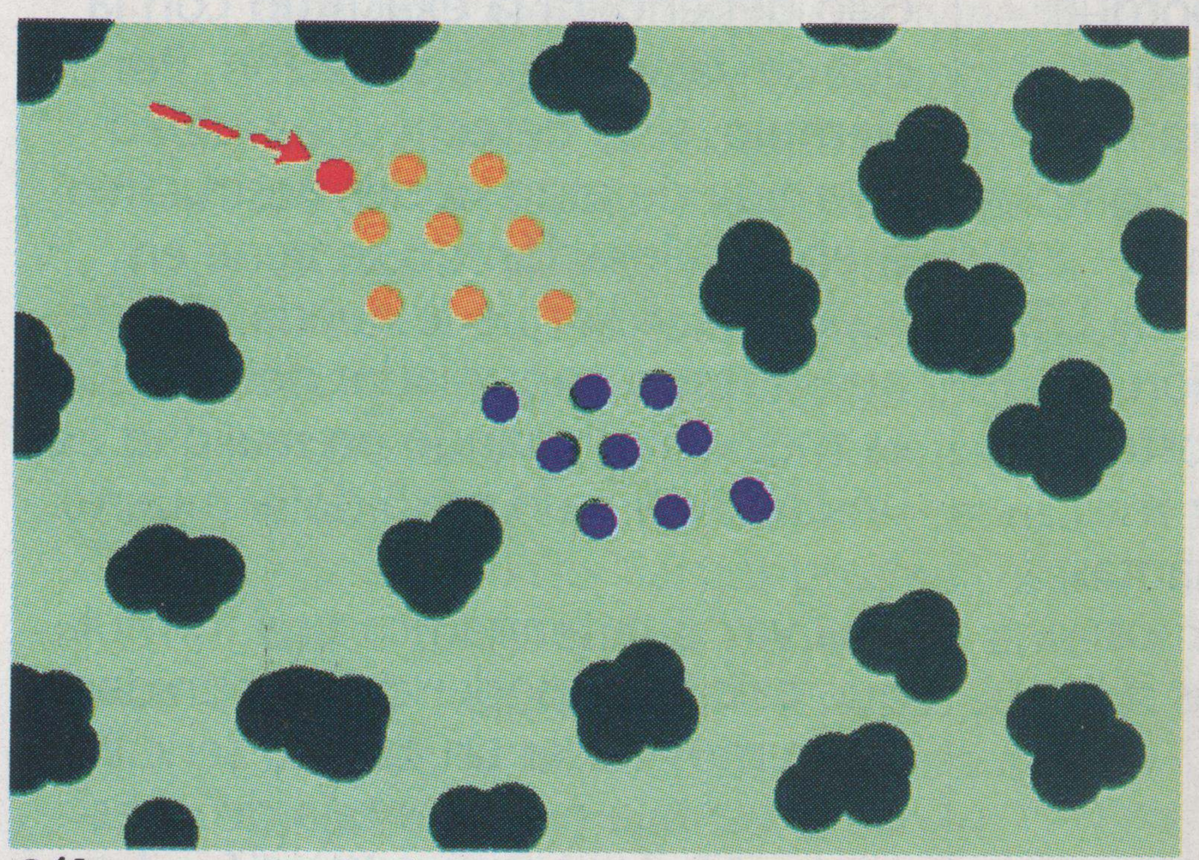
Este cálculo puede realizarse de muchas maneras, dando más importancia a unos valores e ignorando a otros. De la complejidad de este cálculo dependerá la exactitud con la que se valorará una situación. Por ejemplo, si se detecta una unidad enemiga de gran poder destructivo y, en la valoración no se tiene en cuenta la vida que le queda, puede que nuestro controlador decida que es mejor huir cuando en realidad esa unidad está a un paso de la muerte.

Una vez que tengamos una fórmula que calcule el poder de una unidad, podremos hacer un cálculo sencillo de la ventaja o desventaja de nuestras unidades en una situación determinada con tan sólo sumar las valo-

raciones de las unidades propias y restarle las valoraciones enemigas. Si el resultado es positivo, es que tenemos ventaja (cuanto mayor sea el número, mayor ventaja tendremos) y si es negativo, es que tenemos desventaja. Si queremos obtener una medida porcentual de la ventaja, sólo habrá que dividir las sumas de poderes, en vez de restarlas. Si el resultado es mayor de uno tendremos ventaja, en caso contrario, tendremos desventaja. La diferencia estriba en que con la ventaja porcentual, podemos poner un límite a la decisión de atacar independientemente de la suma total de poderes. Por ejemplo, si nos decidimos a atacar, siempre que poseamos una ventaja superior al 25%, lo haremos tanto si somos 126 contra 100, como si somos 126000 contra 100000. En ambos casos, la ventaja es del 26%, pero si el cálculo fuese absoluto, en un caso tendríamos una ventaja de 26 y en el otro de 26000. Según la situación, podremos utilizar una u otra fórmula.

La importancia del cálculo del poder de una unidad

La parte más importante para que este sistema funcione es calcular el poder de una unidad en una situación. Hacerlo demasiado simple puede hacer que el adversario parezca un idiota o un genio. Para poder calcular correctamente el poder de una unidad debemos pensar qué parámetros de los que conocemos afectan a la valoración de la unidad y si afectan de forma directa o inversamente proporcional. Algunos valores que afectan a la valoración de una unidad de forma positiva son la vida actual, la armadura, el poder de ataque, el daño medio que suele infligir, la velocidad, etc. En cambio, afectan negativamente la distancia real hasta poder atacar el objetivo, la separación con respecto al resto de las unidades de mi bando, el terreno, etc. Se puede observar que normalmente los datos independientes de la situación aumentan la valoración, mientras que los dependientes de la situación lo disminuyen.



Sólo acercando la unidad roja a la zona del conflicto mejoramos la situación.

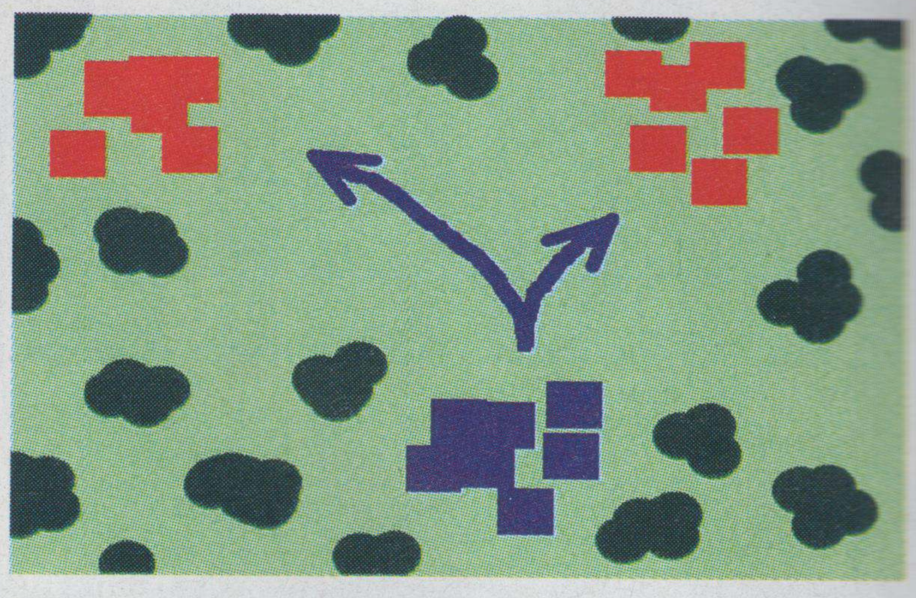
Puede que si en una situación en la no tenemos ventaja modificamos la posición de las unidades, acabemos teniendo ventaja, por lo que es bueno saber cuál es la valoración real y cuál la dependiente de la situación, para saber si podemos mejorar nuestra situación o, si por el contrario, ya está casi optimizada. Conocer los parámetros variables de la formula nos puede ayudar a saber qué es lo que debemos hacer para mejorar nuestra situación. De todas formas, aunque este sistema permite una adaptación al medio relativamente sencilla, no nos ayuda a crear un ejército equilibrado, sino a cómo colocarlo una vez que lo tenemos.

Cómo crear un ejército equilibrado

Para poder saber qué construir y cuándo, debemos preguntarnos lo que solemos hacer como jugadores. Normalmente, se construyen unidades que permiten obtener los recursos necesarios para seguir construyendo otras y luego, se construyen edificaciones que nos permiten construir otros tipos de unidades. Se deben levantar al menos una de cada tipo de unidad, siendo aconsejable construir dos o más de los edificios que nos permiten crear las unidades más comunes y baratas. No debemos construir demasiado lejos del núcleo de población, para que la protección sea más sencilla, ya que programar un controlador es más complicado, sobre todo cuando existe un número elevado de secciones.

Podría ocurrir que para defender una sección se considerase necesario (por los ajustes de poder) enviar las unidades de otra sección, por lo que ésta segunda quedaría desprotegida. Es cierto que esta situación se puede prever, pero es más sencillo limitar la construcción a una determinada zona de terreno, que calcular todas las posibles situaciones. Si se desean varios núcleos, lo mejor es tener varios bandos, cada uno con sus unidades y edificios, como sucede en la mayoría de los juegos de estrategia. Así, cada bando es controlado por un controlador, lo que evita los conflictos.

Siguiendo con el tema de las unidades, lo mejor es predefinir varias tablas con proporciones de los tipos de unidades, según la estrategia a seguir o el enemigo a vencer. Siempre deberemos considerar el ejército como dos partes: la atacante,



Los azules pueden atacar a uno u otro sin que el contrario intervenga para no quedar desprotegido. Por eso no conviene dividirnos.

y la defensiva. Hasta que esta última esté constituida, no deberemos gastar demasiados recursos en la parte atacante. Una vez que la defensa esté constituida, tan sólo deberemos mantenerla, reconstruyendo lo destruido y reponiendo las unidades perdidas. En cambio, el resto de recursos se utilizarán para formar grupos de ataque de mayor o menor potencia. Si nos fijamos en otros juegos del género, el enemigo realiza un ataque cada cierto tiempo con las unidades que haya conseguido producir. De esta forma, se impide al enemigo formar una defensa perfecta y, se le producen bajas que ha de reponer, por lo que no podrá dedicarse de lleno a su objetivo. Se deberá fijar el porcentaje de los recursos que se va a utilizar en la investigación de nuevos tipos de unidades o para mejorar las existentes, pero al ser algo tan subjetivo, tan sólo recordaremos que se debe hacer.

La ventaja de la información

Como decíamos antes, el ordenador basa sus acciones en la información que posee del enemigo, por lo que cuánta más información posea, más inteligente parecerá el adversario. Ésta es una forma de que sea un adversario más difícil, por lo tanto, se le da cierta ventaja sobre su oponente humano, que consiste en que debe conocer todo lo que ocurre en el mapa, aunque no lo vea. Esto le permite preparar estrategias de ataque que de otra forma no podría, pero también puede hacer que las tácticas del jugador no resulten tan efectivas como lo serían contra un jugador humano real. Como segunda opción para mejorar las prestaciones del ordenador, se le pueden dar otras ventajas, además de la información. Un mayor número de unidades o mayores fuentes de recursos, combinados con un pequeño aumento de la velocidad de las unidades recolectoras de recursos, puede hacer que el ordenador resulte un adversario terrible.

Emilio Llamas Alba (YuMoK)



Curso de juegos de rol

Vistiendo y equipando personajes

En el artículo de Divmanía de este mes veremos cómo implementar fácilmente una interfaz gráfica para poder vestir y equipar a los personajes que vayamos metiendo en nuestro juego de rol.

El uso de una interfaz gráfica da una mayor vistosidad al juego y de hecho, todos los juegos de rol actuales las usan.

Las interfaces gráficas, pese a no ser demasiado complejas, presentan ciertos aspectos que las hacen algo más difíciles de hacer que las convencionales interfaces en modo texto. Así que, para ver cómo se podría realizar dicha interfaz gráfica, estudiaremos el siguiente código:

```
Program equipo;

Const

   n_equipado=2;
   n_equipable=5;
   distx=20;
   disty=20;
   startx=250;
   starty=30;

Global

   struct objetos[6]
       string nombre;
       int atack;
       int defense;
       int graph;
       int tipo;
   end

   struct equipadoxy[2]
       int x;
       int y;
   end

   int equipado[n_equipado];
   int equipable[n_equipable];

   int fpg;

Begin

// DEFINIMOS LOS OBJETOS

    objetos[0].nombre="Espada";
    objetos[0].atack=10;
    objetos[0].defense=0;
    objetos[0].graph=1;
    objetos[0].tipo=0;

    objetos[1].nombre="Katana";
    objetos[1].atack=15;
    objetos[1].defense=0;
    objetos[1].graph=2;
    objetos[1].tipo=0;

    objetos[2].nombre="Escudo
peque¤o";
    objetos[2].atack=0;
    objetos[2].defense=10;
    objetos[2].graph=3;
    objetos[2].tipo=1;

    objetos[3].nombre="Escudo gran-
de";
    objetos[3].atack=0;
    objetos[3].defense=15;
    objetos[3].graph=4;
    objetos[3].tipo=1;

    objetos[4].nombre="Armadura de
cuero";
    objetos[4].atack=0;
    objetos[4].defense=20;
    objetos[4].graph=5;
    objetos[4].tipo=2;

    objetos[5].nombre="Armadura de
hierro";
    objetos[5].atack=0;
    objetos[5].defense=30;
    objetos[5].graph=6;
    objetos[5].tipo=2;

    objetos[6].nombre="";
    objetos[6].atack=0;
    objetos[6].defense=0;
    objetos[6].graph=7;

// RELLENAMOS LA ESTRUCTURA
CON LAS COORDENADAS DE LOS
OBJEOS A EQUIPAR

    equipadoxy[0].x=50;
    equipadoxy[0].y=105;

    equipadoxy[1].x=90;
```



```
        equipadoxy[1].y=105;

        equipadoxy[2].x=130;
        equipadoxy[2].y=105;


    // RELLENAMOS LA TABLA EQUIPA-
    BLE CON LOS OBJETOS QUE QUERE-
    MOS

        equipable[0]=0;
        equipable[1]=1;
        equipable[2]=2;
        equipable[3]=3;
        equipable[4]=4;
        equipable[5]=5;

    // OBJETOS EQUIPADOS

        equipado[0]=6;
        equipado[1]=6;
        equipado[2]=6;

    // CARGA DE FICHEROS

        fpg=load_fpg("equipo.fpg");

    // CAMBIAMOS LA RESOLUCION

        set_mode(m320x240);

    // PONIENDO EL FONDO DE PANTA-
    LLA

        put_screen(fpg,20);

    // DEFINIMOS LA REGIÓN QUE
    CONTENDRA LOS OBJETOS EQUIPA-
    BLES Y DIBUJAMOS LAS LÍNEAS PARA
    MARCAR LAS CASILLAS

        define_region(1,startx-
    (distx/2),starty-
    (disty/2),distx*2,disty*((n_equipa-
    ble+1)/2));
        draw(2,34,10,0,240,20,280,80);
        draw(1,34,10,0,260,20,260,80);
        draw(1,34,10,0,240,40,280,40);
        draw(1,34,10,0,240,60,280,60);

    // DEFINIMOS LAS REGIONES DE
    LOS SITIOS DONDE PODEMOS
    EQUIPAR COSAS
        define_region(2,40,95,20,20);
```



```
        draw(2,34,10,0,40,95,60,115);

        define_region(3,80,95,20,20);
        draw(2,34,10,0,80,95,100,115);

        define_region(4,120,95,20,20);

    draw(2,34,10,0,120,95,140,115);

        equipar();


        Loop
            dibujar();

            Frame;
        End


    End

    /////////////////////////////////////////
    ////////////////////////

    Process equipar()

    Private

        byte pos;
        int cobj;
        int cpos;
    Begin
        cpos=-1;
        graph=21;
        Loop
            x=mouse.x;
            y=mouse.y;
            If(mouse.right)
                if(!out_region(id,1))

    pos=halla_pos(mouse.x,mouse.y);

    info_object(equipable[pos]);
                else
                    if(!out_region(id,2))
                        pos=0;
                    end

                    if(!out_region(id,3))
                        pos=1;
                    end

                    if(!out_region(id,4))
                        pos=2;
                    end


    info_object(equipado[pos]);
                end

            End

            If(mouse.left )

                If(!out_region(id,1))

    pos=halla_pos(mouse.x,mouse.y);
                    if(equipable[pos]!=6)
                        if(cpos<0)
```



```
    graph=objetos[equipable[pos]].graph
    ;

    cobj=equipable[pos];
                                equipable[pos]=6;
                                cpos=pos;
                            else
                                graph=21;

    equipable[cpos]=cobj;
                                cpos=-1;
                            end
                        end
                    else
                        If(!out_region(id,2) &&
    cpos>-1 && objetos[cobj].tipo==0)
                            if(equipado[0]==6)
                                equipado[0]=cobj;
                                cpos=-1;
                                graph=21;
                            else

    equipable[cpos]=equipado[0];
                                equipado[0]=cobj;
                                cpos=-1;
                                graph=21;
                            End
                        End

                        If(!out_region(id,3) &&
    cpos>-1 && objetos[cobj].tipo==2)
                            if(equipado[1]==6)
                                equipado[1]=cobj;
                                cpos=-1;
                                graph=21;
                            else

    equipable[cpos]=equipado[1];
                                equipado[1]=cobj;
                                cpos=-1;
                                graph=21;
                            End
                        End

                        If(!out_region(id,4) &&
    cpos>-1 && objetos[cobj].tipo==1)

                            if(equipado[2]==6)
                                equipado[2]=cobj;
                                cpos=-1;
                                graph=21;
                            else

    equipable[cpos]=equipado[2];
                                equipado[2]=cobj;
                                cpos=-1;
                                graph=21;
                            End
                        End
```

```
if(cpos>-1)
        graph=21;

equipable[cpos]=cobj;
        cpos=-1;
      end


      End
    End

    Frame;
  End
End

////////////////////////////////////////
////////////////////////////////////////
///


Function halla_pos(xm,ym)

Private
  int xr;
  int yr;
  byte rx;
  byte ry;
  byte nr;
Begin
  xr=xm-startx+10;
  yr=ym-starty+10;
  rx=xr/distx;
  ry=yr/disty;
  nr=ry*2+rx;
  return(nr);
End


////////////////////////////////////////
////////////////////////////////////////
///


Function dibujar()

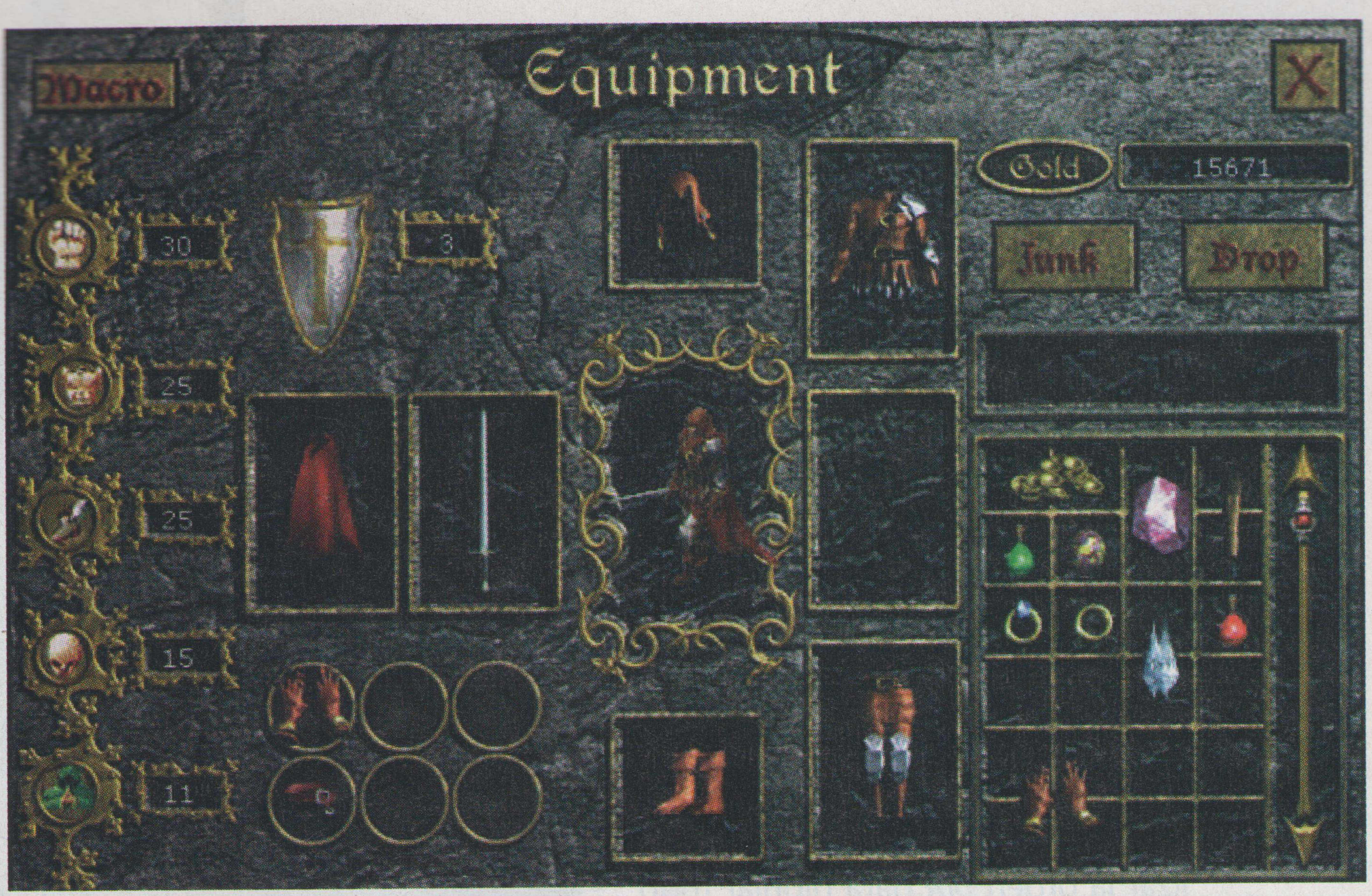
Private

  byte i;
  int temp_map;

Begin


temp_map=new_map(320,240,160,
120,0);
```





Esta es la interfaz gráfica que usa T4C (The 4th coming) para equipar a nuestro personaje.

```
  For(i=0;i=<n_equipable;i+=2)

map_put(fpg,temp_map,objetos[equi
pable[i]].graph,startx,starty+((disty/
2)*i));
  End

  For(i=1;i<=n_equipable;i+=2)

map_put(fpg,temp_map,objetos[equi
pable[i]].graph,startx+distx,starty+((
disty/2)*(i-1)));
  End

  For(i=0;i<=n_equipado;i++)

map_put(fpg,temp_map,objetos[equi
pado[i]].graph,equipadoxy[i].x,equi-
padoxy[i].y);
  End

  put(0,temp_map,160,120);

End


////////////////////////////////////////
//////////////////////////////////

Function info_object(obj)

Private

  int idtext;

Begin

  delete_text(idtext);
  idtext=write(0,200,200,0,obje-
  tos[obj].nombre);

  write(0,200,210,0,"Ataque:");

write_int(0,250,210,0,&objetos[obj].
atack);

  write(0,200,220,0,"Defensa:");

write_int(0,250,220,0,&objetos[obj].
defense);


End
```

Cómo definimos los objetos y las tablas de los arrays que los contendrán

El programa se basa es dos tablas. La tabla *equipados* y la tabla *equipables*. Estas dos tablas contienen un número que indica el tipo de objeto que contienen.

Por ejemplo, *equipados[2]=3* indicaría que en la tercera posición del array encontraremos un objeto de tipo 3.

Los objetos vienen definidos en el array de estructuras *objetos*. En dicho array de estructuras se define el nombre y características que el objeto en cuestión posee. El array de estructuras objetos se define de la siguiente manera:

```
struct objetos[6]
    string nombre;
    int atack;
    int defense;
    int graph;
    int tipo;
end
```

Dentro de la estructura tenemos el nombre del objeto, su ataque, defensa, gráfico y tipo. La variable tipo es usada para distinguir los objetos, ya que, por ejemplo, no es lo mismo una espada que una armadura. La variable graph asocia al objeto el código de un gráfico.

Cómo dibujaremos los gráficos

Dado que son bastantes los objetos que aparecerán dibujados a la vez, no sería demasiado correcto hacer un proceso para cada uno, por lo que habrá que usar un método distinto para dibujarlos.

Para dibujar todos los gráficos, primero crearemos un mapa temporal cuyas dimensiones coincidirán con el alto y ancho de la pantalla (en nuestro caso 320x240). Después, mediante la función *map_put* iremos poniendo los gráficos uno a uno en el mapa temporal que anteriormente hemos creado.

Una vez hemos puesto todos los gráficos tan solo nos hace falta utilizar la orden *put* para dibujar todo el mapa.

La estructura de la función de dibujado sería la siguiente:

```
temp_map=new_map(ancho,alto,an
cho/2,alto/2,color);

    For(i=0;i=<num_obj;i+=2)
                map_put(fpg,destino,ori-
gen,x,y);
    End

put(0,temp_map,ancho/2,alto/2);
```

Cómo averiguar qué objeto se encuentra en unas coordenadas (x,y)

Para averiguar qué objeto se encuentra en una determinada posición utilizaremos la función *halla_pos*, que es la siguiente:

```
Function halla_pos(xm,ym)

Private
    int xr;
    int yr;
    byte rx;
    byte ry;
    byte nr;
Begin
    xr=xm-startx+10;
    yr=ym-starty+10;
    rx=xr/distx;
    ry=yr/disty;
    nr=ry*2+rx;
    return(nr);
End
```



Esta función recibe como parámetros la *x* e *y* donde se pulsa uno de los botones del mouse.

Una vez ha recibido los parámetros, se halla la posición relativa de ese punto respecto al (0,0). Después se busca la región en la que se encuentra (dividimos la zona donde están los objetos en casillas de 20x20 píxels) dividiendo los valores de la *x* y la *y* por el largo y ancho de las casillas.

Una vez ya sabemos la casilla donde se encuentra, encontramos qué objeto es mediante *nr=ry*2+rx*, ya que el número de casillas horizontales siempre es 2.

Uso de las regiones

En este ejemplo se hace un uso extensivo de las regiones, que son de gran utilidad para determinar si un proceso se encuentra en una zona rectangular previamente determinada.

Para usar una región en DIV, primero se ha de crear la región con la función:

```
define_region(<numero de
región>,x,y,ancho,alto);
```

Una vez que la región está definida, tan sólo hemos de usar la función:

```
out_region(<id de
proceso>,<numero de region>);
```

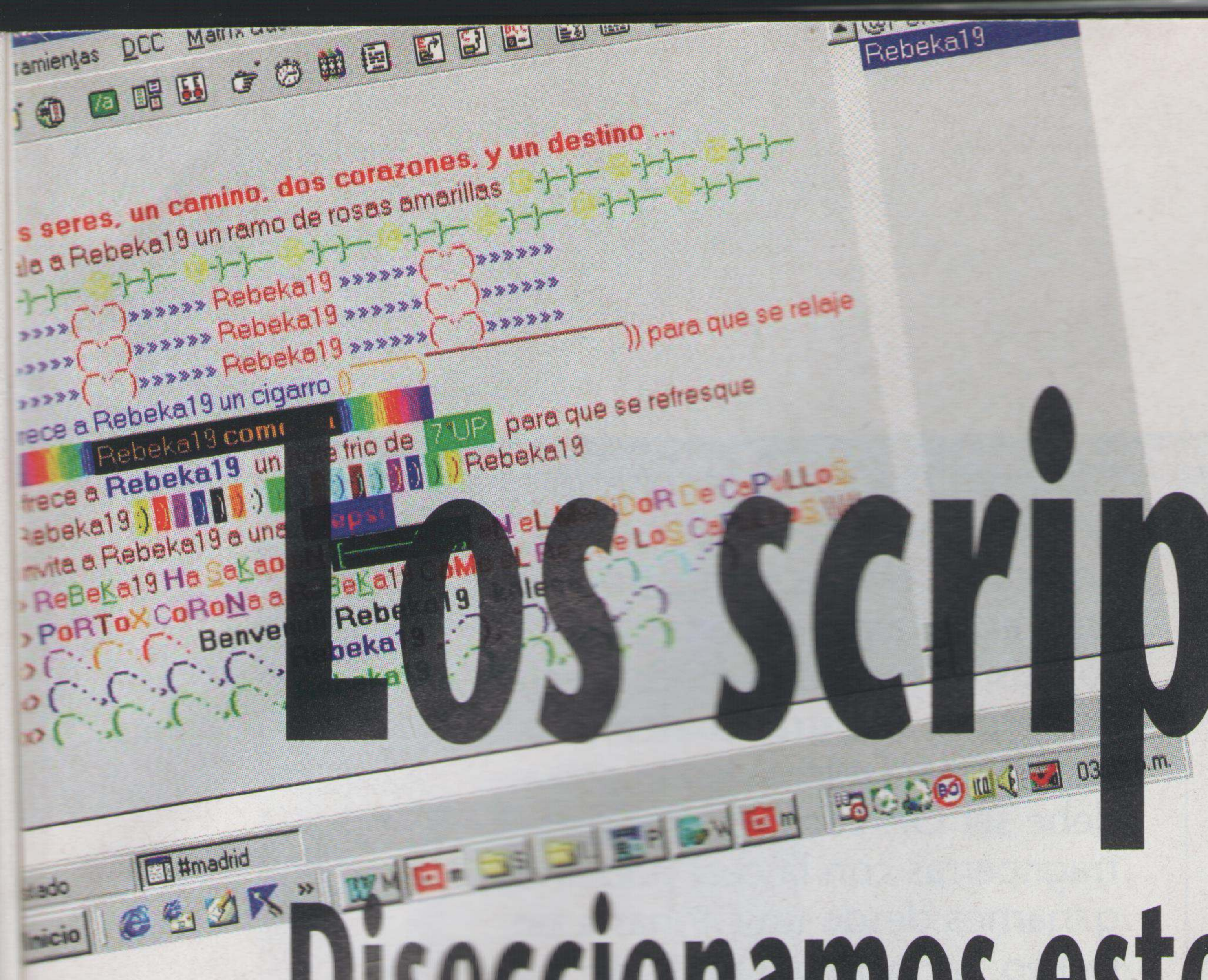
para comprobar si un proceso está dentro o fuera de la región.

En nuestro ejemplo, se usa para saber si el proceso a equipar se encuentra en una determinada zona y así poder actuar en consecuencia. Por ejemplo, si se encuentra en la región 1 sabemos que se trata del lugar donde están los objetos equipables, si es la región 2 es donde equiparemos la espada, etc.

Ramón de España
maqnaziller@pagina.de



Captura del menú de equipo de Asheron's Answer.



Los scripts

Diseccionamos estos programas

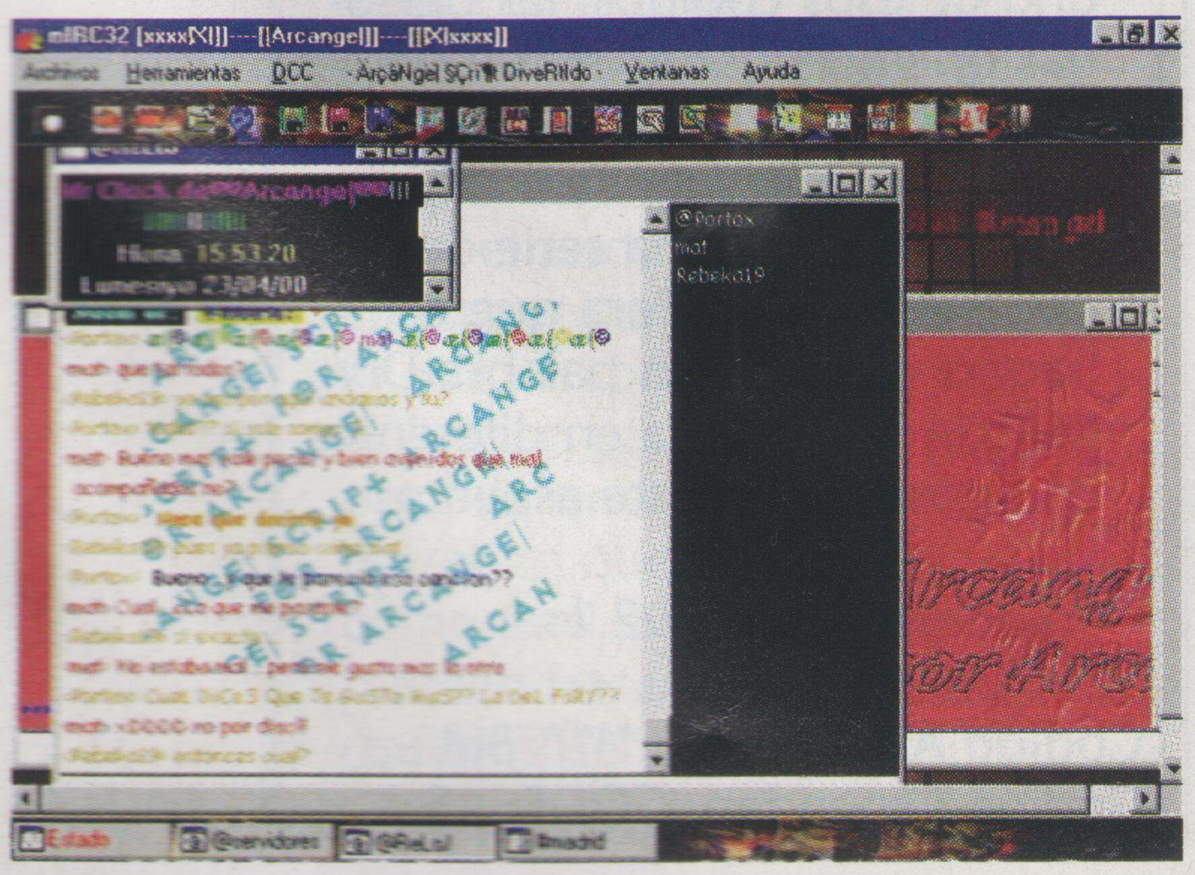
Los Scripts son programas para el IRC (Internet Relay Chat), es decir, que sirven para chatear. Estos Scripts están basados principalmente en un programa, el famoso mIRC, a partir de él y mediante una serie de menús llamados Alias, Popus y Remotos que se pueden crear fácilmente, surge lo que se conoce como Script.

Resumiendo, que un Script es el programa mIRC modificado por una o más personas, al añadirle nuevos menús y nuevas opciones.

Tipos de scripts

Los Scripts existentes actualmente, tanto en español como en castellano son muchos y muy variados, los hay que están orientados principalmente para atacar a otros usuarios del IRC molestándolos o desconectándolos, lo que se conoce como IRC WAR, los hay para defenderse de estos ataques, los hay orientados simplemente para hacernos más ameno el chateo incluyendo opciones muy interesantes como escribir con colorines, contar chistes, jugar a juegos, etc. También los hay orientados hacia principiantes, aunque la mayoría de los Scripts incluyen una mezcla de todas estas opciones.

Los Scripts nos protegen del ataque de otros usuarios en lo que se ha llamado la IRC-WAR



Principales aplicaciones de los scripts

Es muy difícil ponerse de acuerdo en qué aplicaciones extras nos puede aportar un Script a nuestro chateo, ya que como dijimos antes eso depende mucho del tipo de Script, aunque intentaremos mostraros la mayoría de las aplicaciones extras que existen, un Script no tiene por que tenerlas todas.

Para empezar hay que decir que aunque el Script es una modificación del mIRC, incluye todas las opciones que trae de por sí este programa, lo que quiere decir que por lo pronto ya tenemos una gran lista de servidores donde conectarnos, la posibilidad de grabar nuestras conversaciones en ficheros de texto, más conocidos como Logs, la posibilidad de enviar y recibir archivos mediante DCC o de chatear por DCC evitando el fatídico LAG y muchas otras opciones más.

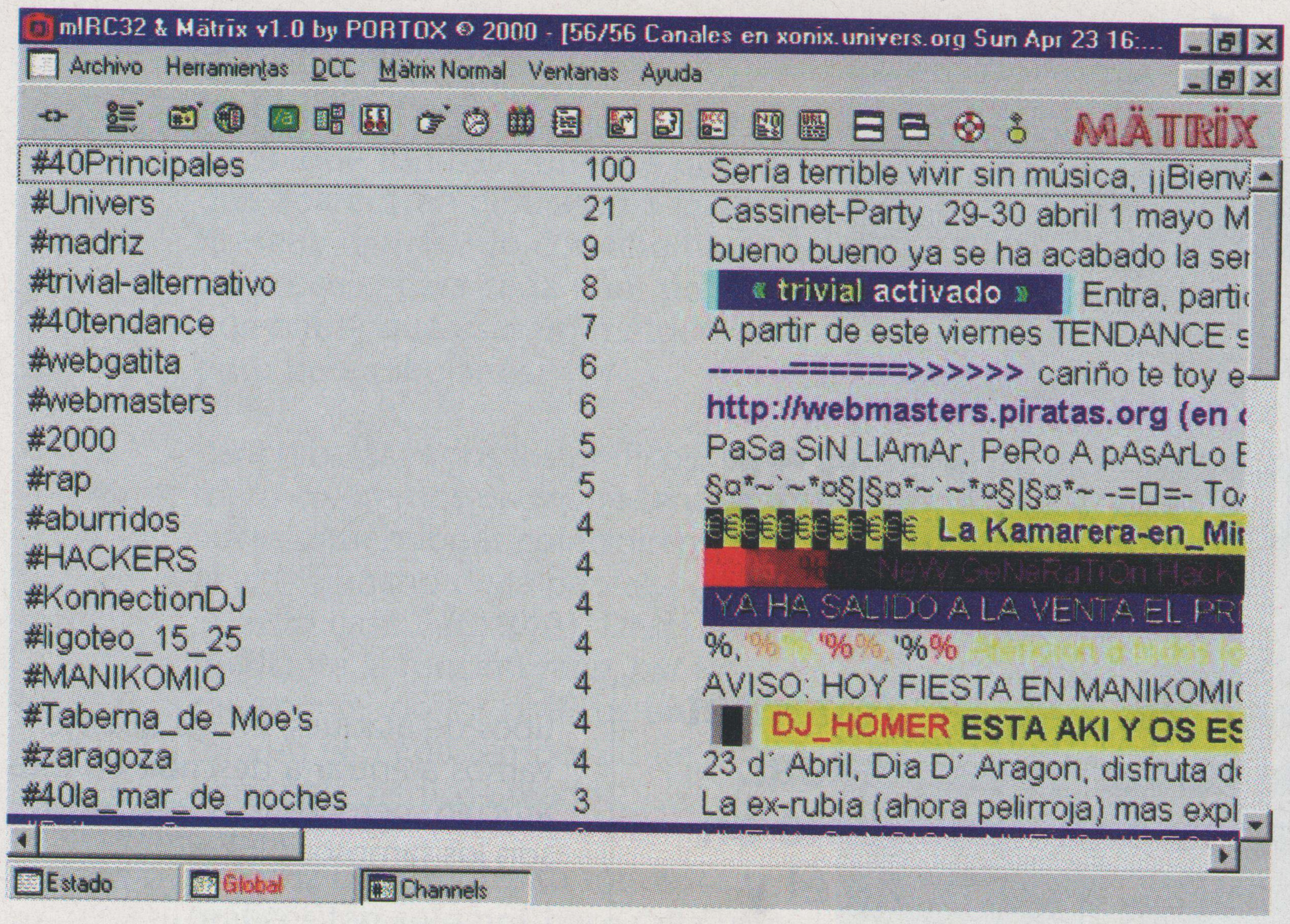
Pero, como muchas veces esas opciones no son suficientes, los Scripts incluyen muchas otras que hacen mas interesante y ameno el chateo, a continuación explicaré muchas de ellas:

- IRC-WAR: Como ya explicamos antes, esta opción sirve para poder atacar a los otros usuarios del IRC molestándolos o desconectándolos, raro es el Script que hoy en día no incluya al menos alguna que otra opción de IRC-WAR, dentro de la IRC-WAR existen muchos tipos de ataques, en los que no vamos a entrar a describir en este artículo, pero los mas conocidos son los famosos Nukes.
- PROTECCION: Al igual que existe la IRC-WAR también existen programas para protegerse de estos ataques, hoy en día todos los Scripts llevan protecciones de este tipo en mayor o menor número y eficacia, protecciones contra Nukes, Flood y otros tipos de ataques son casi imprescindibles para que nadie nos pueda molestar en el IRC.
- CLONES Y BOTS: Son copias de tu mIRC que cargas en el servidor y hacen que estés presente en el IRC varias veces. Se pueden usar para atacar a otros usuarios mediante flood o bien si tienes Op, puedes cargar un bot y hacerle Op, así si te tiran, aun conservarás una copia tuya con Op. Tradicionalmente los Clones son las copias que se usan en la IRC-WAR y los Bots son las que se usan para fines pacíficos.
- OPCIONES OP: El mIRC incluye de por sí ya opciones para cuando

mIRC

Es un programa muy utilizado por lo internautas para chatear por la red debido a sus amplias posibilidades para modificarlo a nuestro gusto cambiando los menús, así como por su sencillez y rapidez. El programa lleva ya varias versiones y la primera salió a la luz en 1995. Si queréis mas información o bajaros el programa visitar http://www.mirc.co.uk y si queréis aprender a usarlo os aconsejo que visitéis:

http://www.readysoft.es/home/ihidalgo/contenido.html

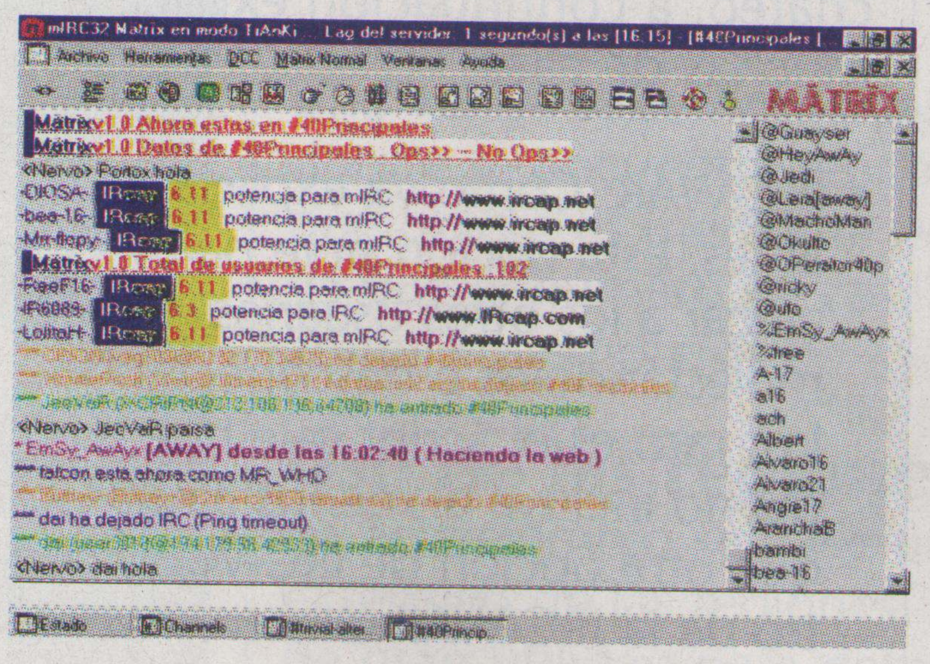


seamos OP, como kickear o banear, pero estas opciones se ven incrementadas en los Scripts, ya que incluyen opciones como hacer OP´s masivos o kicks y bans masivos, o kicks con frases divertidas. También podemos activar protecciones para que el Script kickee automáticamente a aquellos usuarios del IRC que estén molestando, de esta manera no tendremos que estar vigilando quién molesta o no para echarlo, el Script lo hará automáticamente por nosotros así podemos hacer que el Script kickee automáticamente a quien escriba palabrotas, quien tenga clones, quien flodee el canal escribiendo muchas frases seguidas, quien escriba con mayúsculas y muchas otras opciones mas.

Otras opciones

Pero no os penséis que todas las opciones extras de los Scripts están dedicadas íntegramente a atacar y defenderse de los ataques de otros usuarios, como os mostraremos a continuación existen una inmensa cantidad de opciones que no están pensadas mas que para amenizar el rato que pasemos en el IRC:

- TEXTOS ESPECIALES: Con este nombre designamos las opciones que tienen muchos Scripts para poder escribir lo que deseemos con letras y colores especiales, así, podemos escribir textos con letras gigantes, con los colores de nuestro equipo favorito, con sombras, con marcos, en bloques , y con muchos otros estilos.
- MENSAJES ESPECIALES: En este apartado se incluyen las opciones que tienen muchos Scripts para poder dedicar mensajes especiales de bienvenida, despedida, de amor, de amistad, de acciones y otros muchos más, a un nick.
- DIBUJOS: Los dibujos son archivos TXT , formados mediante puntos, guiones, barras, letras u otros signos, que debido a la unión de estos, forman diferentes dibujos que pueden ser en blanco y negro o en color y que se muestran al canal, existen multitud de dibujos, algunos muy curiosos, que van desde objetos hasta animales o personas.
- JUEGOS: Existen distintos juegos para entretenernos con otros usuarios del IRC mientras chateamos, nos podemos encontrar desde el famoso tetris, hasta el ahorcado, el cara y cruz (muy útil para echar algo a suertes) e incluso existen juegos que nos afectan de verdad mientras chateamos, aunque para ello quien lo active debe ser OP, como la maquina tragaperras con la cual podemos ganarnos desde un OP hasta un kickeo, un baneo o un Nuke y la famosa ruleta rusa, con la cual, al que se le dispare el arma será kickeado automáticamente.



- SONIDOS: Otra opción interesante es la referente a los sonidos, muchos scripts incluyen opciones especiales referentes al tema, como archivos MIDI o MP3 que podemos escuchar mientras chateamos, además podemos reproducirlos a todo el canal y no sólo a nosotros, para que también lo oigan los demás usuarios del IRC y por si esto no fuera suficiente también se puede disponer de la opción de escuchar la radio en directo a través de Internet, para que no nos podamos aburrir en ningún momento mientras estemos en el IRC.

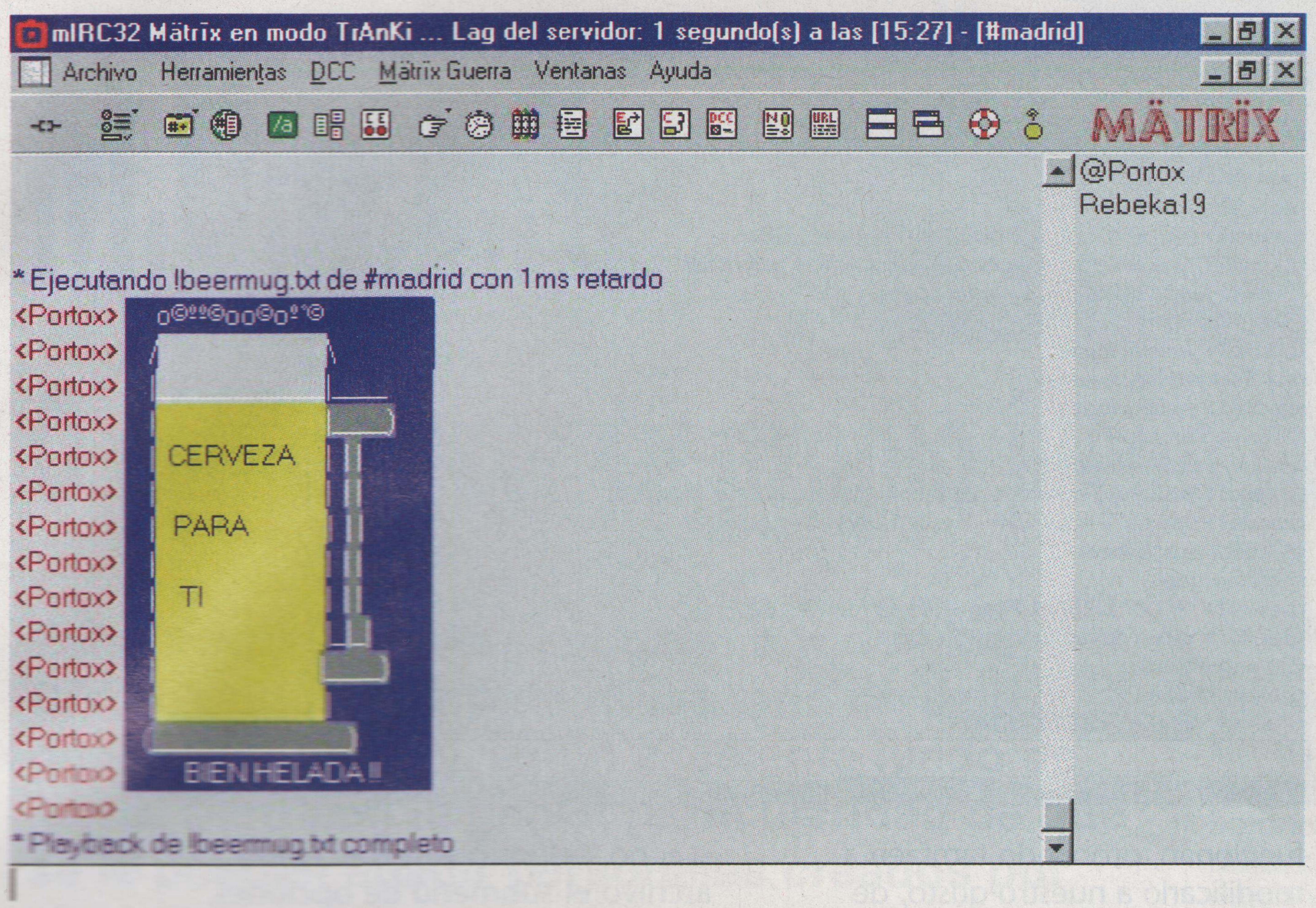
Existen distintos juegos para entretenernos con otros usuarios del IRC mientras chateamos

- RELOJ: El reloj es una opción muy corriente, aunque existen varios tipos, los hay que simplemente te muestran la hora y los hay que nos dicen el tiempo exacto de conexión e incluso algunos el gasto en pesetas que llevamos efectuado desde que nos conectamos, muy útiles para que la factura de telefónica no nos sorprenda a final de mes.
- ENCRIPTACIÓN: La encriptación es un método utilizado en el IRC para transmitir mensajes codificados y sirve para que nadie mas que los que tengan la clave de descodificación puedan leerlo, existen muchos métodos de encriptación con códigos diferentes, de manera que si escribimos un mensaje y lo encriptamos en uno de esos códigos, las personas que no posean el código con el que hemos encriptado o cifrado el mensaje solo verán aparecer en pantalla una serie de extraños signos. Para que os hagáis una idea, la palabra DIV-MANIA encriptada en diferentes códigos tendría este aspecto:

Código ScriPtteR´s v2.1: ïÆ•¢©¥Æ©
Código AZZ-CODE: *ª″^¨%ª¨
Código Legendary CODE: ~§

¿Qué es el LAG?

Es la cantidad de tiempo en segundos entre lo que un usuario escribe y otro u otros leen lo escrito. Es decir, si hay dos segundos de lag entre 'A' y 'B' entonces lo que A escriba, B no lo va a ver hasta dentro de dos segundos después. Por lo tanto cuanto menos lag haya entre los usuarios, más ágil y rápida será la comunicación entre ellos.



...¶¶¶¶¶¶¶¶
Código 667 CODE: \´ç%¡_´¡
Código Total Eclipse CODE:
ôª__@ñª@ - ITExENI

Como podéis observar, para cualquier persona que no posea el código de desencriptación le sería imposible conocer la palabra que hemos escrito ya que sólo aparecen signos sin sentido. Este método es muy útil para comunicarnos a través del IRC con otra persona sin temor a que alguien pueda estar leyendo lo que escribimos.

Hay posibilidad de codificar nuestra conversación con una persona para que otros usuarios no la lean

- ALERTA DE NICK: Algunos scripts incluyen esta interesante opción que tiene la finalidad de saber si alguien está hablando de nosotros o a nosotros cuando no nos encontramos presentes. Cuando nos encontramos en varios canales es muy difícil poder atender a todos a la vez y hay veces que si nos dicen algo no nos enteramos por que estamos en otro canal chateando, con esta opción, en cuanto alguien escriba nuestro nick en un canal, ya sea para llamarnos, para comunicarnos algo o, aprovechando que no nos encontramos presentes, para meterse con nosotros, el script nos avisará automáticamente.

- SISTEMA AWAY: Hay veces que por determinadas razones no podemos atender a un canal, bien por que estamos muy ocupados chateando en otro, porque estamos navegando por Internet, por que estamos muy ocupados con un privado o simplemente porque tenemos que ausentarnos un momento del ordenador. Este sistema tiene la función de un contestador automático, cuando una persona se pone AWAY, es decir, activa el sistema AWAY, automáticamente si se desea su nick cambia y pasa a aparecer en él la palabra AWAY al final, para advertir a cualquier usuario que no se encuentra presente. Aparte de eso se puede hacer que cada cierto tiempo el script haga aparecer una frase en la que se comunique al canal que nos encontramos ausentes y la razón por la que nos encontramos ausentes. Aún así, si alguien nos escribe un mensaje, queda grabado para que cuando volvamos podamos leerlo y además, a la persona que nos envió el mensaje, automáticamente se le devuelve otro advirtiéndole que nos encontramos ausentes pero que su mensaje ha quedado grabado y que se le contestará en cuanto sea leído.

¿Qué es un OP?

Es el usuario encargado de velar por el orden y mantenimiento del canal. Se identifican por la arroba "@" que tienen junto a su nombre y tiene el poder para echar a cualquier persona del canal, lo que se conoce como kick o kickear, o para echarla sin permitir que pueda volver a entrar, lo que se conoce como ban o banear. También pueden hacer OP a otros usuarios del canal.

Algunos scripts

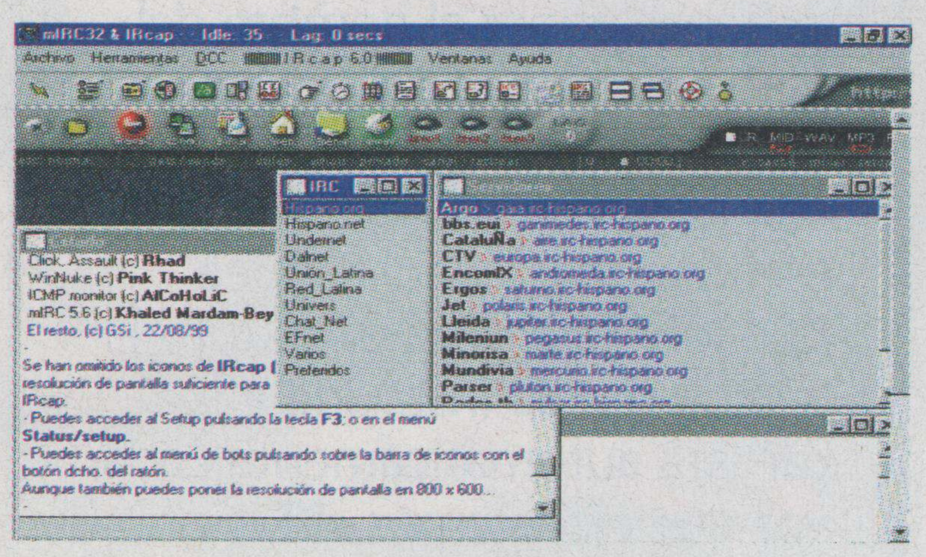
Ya hemos hablado de casi todas las características que incluyen los Scripts, pero a la hora de elegir qué Script usar cada usuario tiene que elegir el que mejor se adapte a sus necesidades, un Script muy utilizado y famoso es el IRCap debido a su interfaz gráfica y a que posee un menú muy cuidado desde el que podemos configurar todo el Script, otro también muy famoso y utilizado es el Orbital Script que incluye más opciones que el primero pero no es tan fácil de configurar y usar.

Estos son los Scripts más utilizados, de habla hispana claro, por que no debemos olvidar que existen también muchos Scripts extranjeros y no por ello menos importantes, como por ejemplo 7Th Sphere que es el Script extranjero usado por excelencia y además es muy bueno. Si deseáis buscar un Script a vuestra medida os aconsejo que visitéis http://www.guitarra.net/irc/scripts.htm. En esta pagina se encuentran la mayoría de los Scripts existentes ordenados alfabéticamente, además se incluye una pequeña descripción de cada Script para saber si nos interesa o no, en esta pagina podéis encontrar Scripts desde para los más novatos hasta para los más expertos.

Parches y accesorios

Los parches son pequeños programas para mIRC que lo que hacen es añadir nuevas propiedades al mIRC o al Script, son muy útiles para añadir nuevas e interesantes opciones al Script que estemos utilizando, así tenemos parches que aumentan la velocidad en el envío de archivos, que nos alertan cuando una palabra que hayamos configurado previamente sea nombrada en el canal e incluso los hay que son capaces de mantener una conversación con otros usuarios del IRC sin que nosotros tengamos que intervenir para nada (muy útil si alguien no para de hablarnos y no tenemos ganas de hablar con él).

Los accesorios en cambio son programas individuales que, aunque funcionan sin estar usando el Script, aportan nuevas propiedades a éste, algunos accesorios pueden



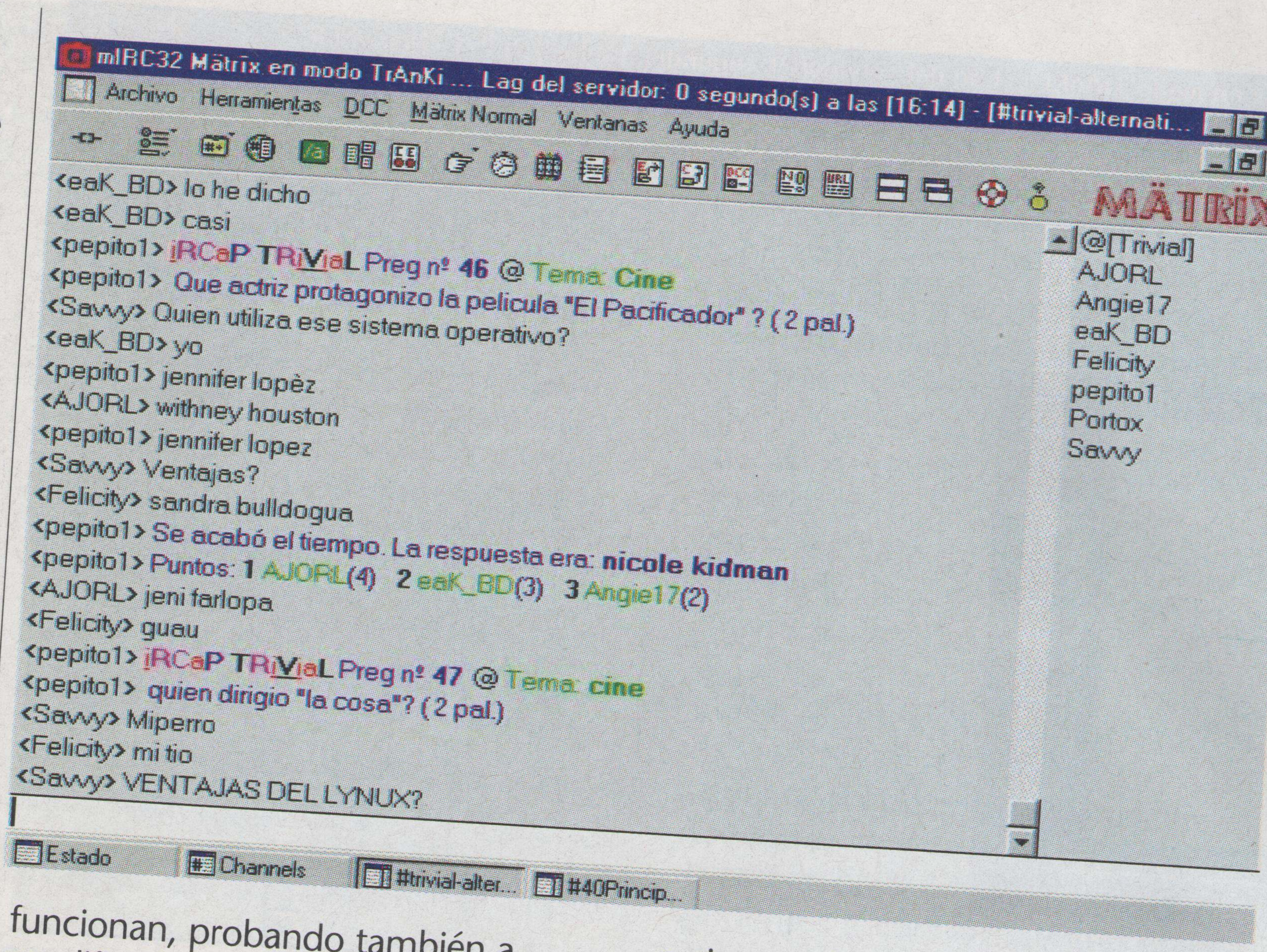
ser programas para escribir con colores, Mail-Bombings, armas de ataque, programas para conectarse anónimamente, etc.

Una buena pagina cargada de Parches y Accesorios es http://www.guitarra.net/irc/parches.htm Desde aquí podréis bajaros los últimos parches y accesorios para el mIRC.

Scriptting

Se conoce con este nombre a la técnica de programar o crear Scripts, así como el lenguaje utilizado para crearlos. Aprender a crear tu propio Script no es una tarea difícil si tenemos en cuenta que el lenguaje utilizado es muy simple y que sólo nos hace falta un editor de textos. Simplemente hay que saber que existen tres menús que tenemos que crear, los Alias, los Popups y los Remotos. También tenemos que crear las variables que utilizaremos en nuestro programa.

El tema de Scriptting es demasiado largo debido al innumerable número de sentencias y combinaciones, por lo que no podemos desarrollar el tema mucho mas, pero si os interesa existe una pagina muy buena que explica muy bien paso a paso como crear tu propio Script la dirección es: http://www.guitarra.net/irc/scriptting.htm y si tenéis alguna duda podéis escribirme y estaré encantado de ayudaros. Si aún así no sabéis bien cómo empezar, lo mejor es bajarse un Script ya hecho para estudiarlo y comprender mejor cómo



funcionan, probando también a modificarlo a nuestro gusto, de hecho, la mayoría de los Scripts están basados en otros previos.

Razones por las que usar un Script

Ya hemos hablado de la cantidad de opciones que puede incluir y tener un Script, pero también hay que advertir que, para un usuario poco avanzado en el campo de la informática o en Internet, puede costarle empezar a manejar un Script, aunque sin lugar a dudas os animamos a que los uséis debido a que aporta mucha más comodidad a la hora de chatear y sobretodo mucha más rapidez que hacerlo desde una pagina Web.

Un punto que puede resultar negativo a la hora de usar un Script es que no existen muchos servidores de habla hispana por lo que tendremos que añadirlos nosotros. Pero eso no es ningún problema acontinuación os escribimos algunos de los servidores mas conocidos e importantes de habla hispana y sobre todo de España junto con el puerto de conexión:

irc.cadena40.es 6667 | Chat5.telecinco.es 6667 | irc.catalunya.net 6666
pegasus.irc-hispano.org 6667 | es.unionlatina.org 6667 | irc.arrakis.es 6664 | interlink.es.unionlatina.org 6667 | snemeis.univers.org 6667 |
irc.milenium.com 6667 | namek.univers.org 6667 | irc.jet.es 6667 | irc.irc-hispano.net 6667 | irc.irc-hispano.org 6667 | irc.satlink.com 6667

Para añadir estos nuevos servidores al Script tenéis que pulsar Alt+O una vez ejecutado el Script, o si no, seleccionar en el menú archivo el submenú de opciones, seleccionar la categoría Conectar y pulsar en el botón Añadir, una vez hecho esto, en la casilla Descripción escribir el nombre que queráis para reconocer el servidor, en Servidor IRC escribir el nombre del servidor que queráis añadir, en Puerto escribir el número de puerto que aparece a la derecha del servidor que habéis añadido y después pulsar el botón añadir. Ya hemos añadido el nuevo servidor ahora bastará seleccionarlo y pulsar en el botón Conectar Servidor IRC para conectarnos a él.

Si tenéis alguna pregunta o duda de cómo usar un Script o queréis aprender a crearlos, sólo tenéis que escribirme un e-mail y yo estaré encantado de ayudaros.

Daniel García Alonso (PORTOX)
Dagal@Eresmas.com

¿Cómo instalar un parche?

Los parches no son como los accesorios que son programas que se pueden ejecutar directamente, los parches son archivos con extensión *.mrc que para que se instalen en el mIRC o el Script han de ser cargados previamente, para ello debéis copiar el parche al directorio donde tengáis instalado el Script, luego ejecutar el Script y escribir: /load -rs parche.mrc entendiéndose que en parche.mrc deberéis poner el nombre del parche a instalar. Así, si por ejemplo fuéramos a instalar un parche llamado speedup.mrc escribiríamos /load -rs speedup.mrc una vez hecho esto el parche queda instalado y se cargará automáticamente cada vez que ejecutemos el Script.

¿Qué es exactamente un Servidor?

Un servidor es el lugar donde nos podemos conectar para chatear, cada servidor esta formado por un número determinado de canales y cada usuario que se conecta a él tiene la posibilidad de entrar a cualquiera de estos canales o de crear otros nuevos. Para hacernos una idea rápida, digamos que cada servidor es como un edificio y cada canal de ese servidor es como cada habitación de ese edificio de manera que cada cual se puede meter en la habitación o canal que más le interese de ese edificio o servidor.

M.U.G.E.N.



O cómo hacer tu propio juego de lucha

Este programa está destinado a todos los DIVeros que seáis auténticos fans de este género. M.U.G.E.N. es un excelente juego de lucha, pero con la genial particularidad de que se le pueden añadir personajes creados por uno mismo o descargados de Internet.

De esta manera se ha convertido en el juego con más luchadores de la historia, y en la red podéis encontrar personajes de todas las sagas, desde Ryu o Athena a Megamán.

Este *engine* de juegos de lucha programado por el grupo Elecbyte nos permite crear personajes a nuestro gusto, sin otra limitación que nuestra imaginación y capacidad técnica. Podremos crear desde luchadores sencillos a los más complejos con ataques especiales, *supers*, proyectiles, *fatalities* e incluso con otros luchadores o animales como ayudantes. Asimismo podremos crear escenarios con su música de fondo y añadirlos al programa. Pero lo más atractivo de todo es la posibilidad de enfrentar nuestros luchadores a otros creados por otras personas, en un plantel inmenso que incluye a los luchadores más famosos de la historia de los videojuegos.

De hecho, la modalidad más popular es la de convertir personajes de otros juegos de lucha a M.U.G.E.N. Los juegos más usuales para esta conversión son los *King of Fighters* y los *Street Fighter*, y en Internet podemos encontrar a casi todos los luchadores de estos juegos. Instalarlos para poder jugar con ellos es tan sencillo como descomprimirlos en un directorio propio de *mugen\chars* y añadirlos a la lista de personajes que hay en *mugen\data\select.cfg*.
Aún así, debo advertiros que crear un luchador para M.U.G.E.N., así como convertirlo de otros juegos, es una tarea ardua, que os llevará varios días y os producirá algún que otro dolor de cabeza. El programa está aún en fase beta (aunque es plenamente funcional), y entre otras cosas la documentación sobre cómo crear luchadores es bastante escasa. Pero en este artículo hallaréis lo fundamental para iniciaros.

Para empezar hay que decir que para jugar sólo necesitáis el programa principal, pero para crear luchadores necesitaréis más cosas. Son imprescindibles las *M.U.G.E.N. Tools*, creadas también por Elecbyte. Asimismo es más que recomendable que para empezar utilicéis el luchador básico, que encontraréis en *player.zip* y que contiene los archivos mínimos, así como explicaciones sobre su funcionamiento. También resulta muy útil un programa llamado *Mugen Character Maker*, programado por un español, que simplifica bastante la creación de personajes. Todo esto lo encontraréis en el CD de DIVManía.

Otra cosa que necesitáis (aparte de paciencia) es un bloc de notas para apuntar la larga lista de *sprites*, animaciones y movimientos del personaje. Si no lo hacéis será mucho más difícil cambiar cosas después.

Los archivos necesarios

Como ya he dicho, los archivos que definen cada luchador están en un subdirectorio de *chars*. Hay varios archivos pero básicamente se reducen a tres clases: los ficheros de gráficos y sonido, que se crean con dos programas auxiliares; los archivos con las paletas, es decir, los colores de los luchadores; y finalmente, toda una serie de ficheros de texto que definen el comportamiento del luchador, desde las animaciones que puede hacer en cada momento a los botones que deben pulsarse para realizar un ataque especial. Los veremos ahora uno por uno.

Supongamos que nuestro luchador se llama *luchador*. Así pues, deberemos crear un directorio *mugen\chars\luchador* y en él deben estar los siguientes archivos:

-***luchador.def***: Este es el primer archivo que necesita el programa, pues contiene la información básica: el nombre del luchador y los archivos que lo componen. Lo podéis crear en un editor de texto cualquiera, modificando el archivo *player.def*.

-***luchador.sff***: Es el fichero de gráficos, que contiene todos los *sprites* del personaje (algo así como los *.fpg* del DIV). Para crearlo deberemos utilizar el programa Sprmaker que viene con las *Tools*. Este programa creará el *.sff* a partir de ficheros *.pcx*, que podréis crear con cualquier editor gráfico. Deberemos organizar los *sprites* por grupos, para que sea más fácil utilizarlos después. Además deberemos introducir las coordenadas del punto base del luchador, que está centrado y en el suelo (ver Fig. 1).

Para que introducir todos esos datos no se haga pesado, se puede automatizar el proceso con

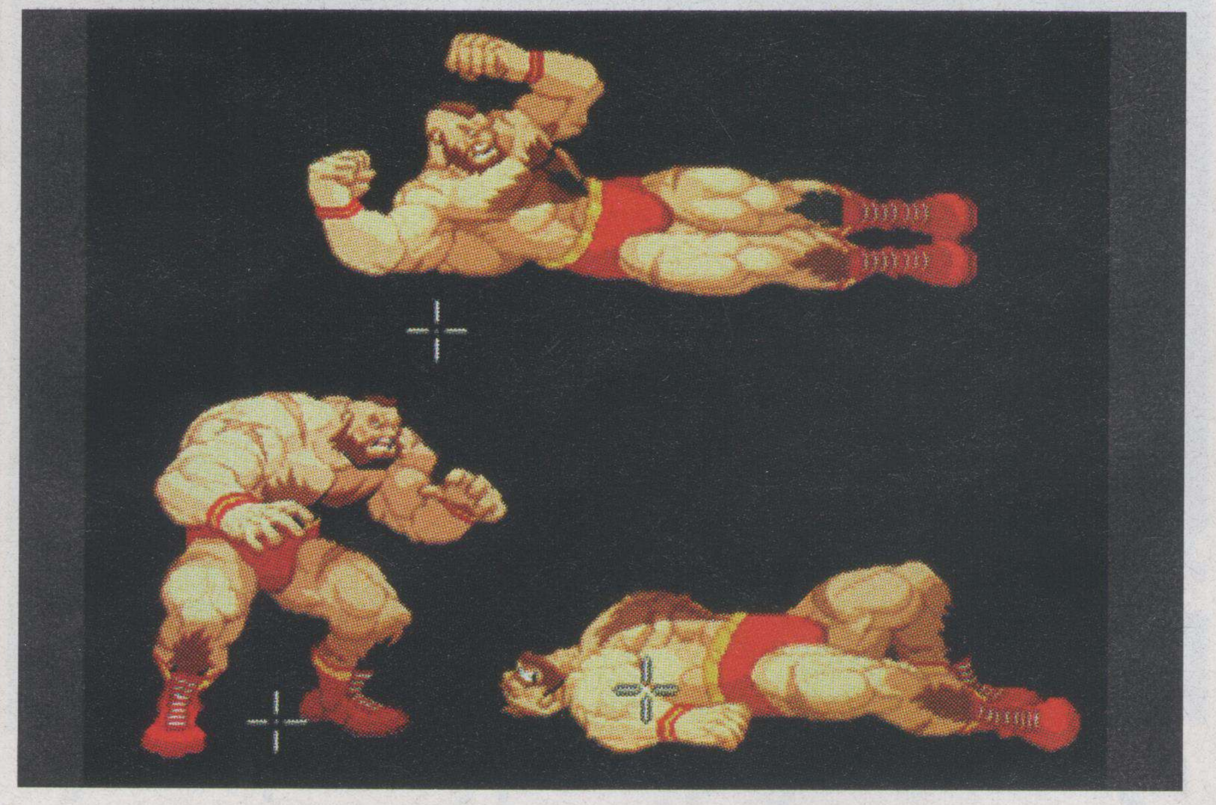


Fig. 1: Posiciones correctas del punto base.

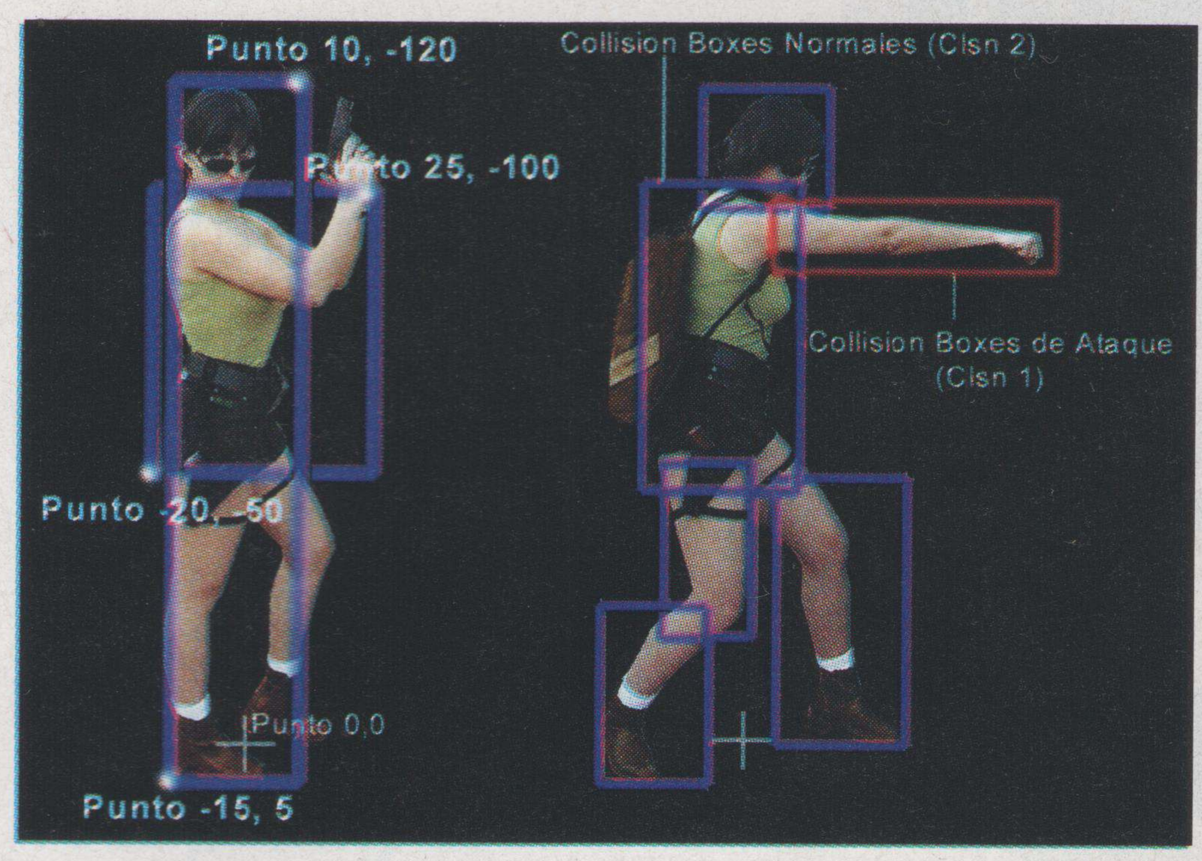


Fig. 3: Collision Boxes normales y de ataque.

un fichero de texto. Encontraréis las instrucciones para hacer esto en el *readme.txt* de las *Tools*. Otra opción que seguramente os resultará más sencilla es usar el mencionado programa *Mugen Character Maker*, que hace la conversión mediante un práctico *interface* visual. (ver Fig. 2).

Nota importante: Los luchadores está limitados a una paleta de 8 bits, y todos los sprites deben tener la misma paleta. Esto no debe resultar una gran limitación, y de hecho con DIV ocurre lo mismo.

• ***luchador.snd***: Es el fichero de sonidos, que se crea de manera muy similar al *.sff*, pero partiendo de ficheros *.wav* y utilizando el programa Sndmaker de las *Tools*. Hay que decir que este fichero no es imprescindible, pero si no lo incluimos nuestro luchador no tendrá sonidos.

Pueden participar hasta cuatro luchadores simultáneos, sin contar los helpers

• ***luchador1.act, luchador2.act, etc.***: Estos ficheros también son opcionales, y contienen las distintas paletas que tendrá el luchador en función de con qué botón lo seleccionemos en el juego. Se pueden crear con Photoshop o cualquier otro programa que pueda salvar paletas con extensión *.act*. Si no los incluimos el luchador sólo tendrá una paleta, la del fichero *.sff*.

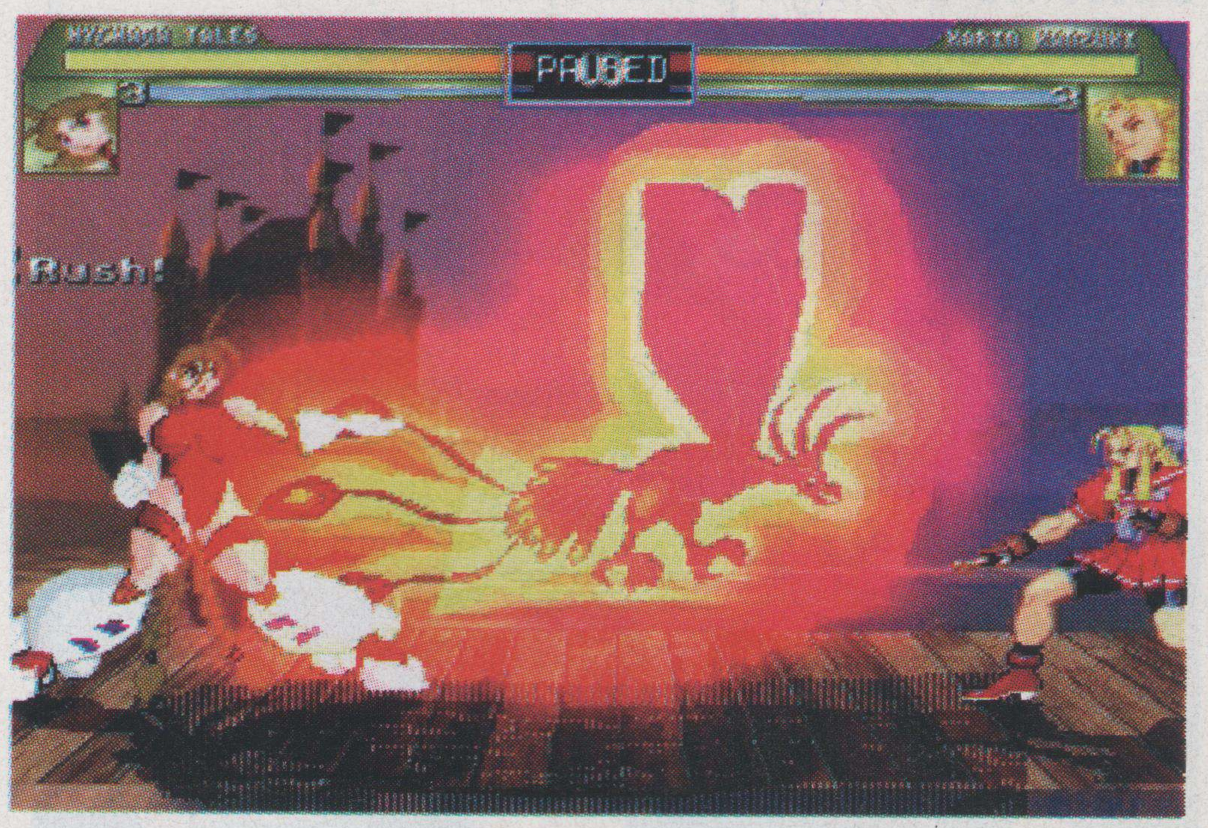


Fig. 4: Los efectos de transparencia permiten crear impresionantes ataques.

Ahora veremos los ficheros de texto que marcan el comportamiento del luchador. Para crearlos sólo necesitáis un editor de texto. Lo mejor es que modifiquéis los que vienen en *player.zip* y les cambiéis el nombre por el de vuestro personaje.

• ***luchador.air***: Aquí se describen dos cosas: las animaciones del personaje y sus *collision boxes,* que no son otra cosa que rectángulos que describen su forma. Los *boxes* de tipo 1 definen las zonas con las que el luchador puede golpear, y los de tipo 2 las zonas donde puede ser golpeado (ver Fig. 3).

Cada animación y sus correspondientes *boxes* se incluyen en una acción. Una acción es una lista de sprites (procedentes del fichero *.sff*) con sus *collision boxes* correspondientes. Veamos esto con un ejemplo:

```
;Posición Inicial
[Begin Action 0]
Clsn2Default: 2
 Clsn2[0] = -15,5,10,-120
 Clsn2[1] = -20,-50,25,-100
0,1,0,0,30
0,2,0,0,30
0,3,0,0,30
0,2,0,0,30
```

Esta es una de las acciones de un fichero *.air*. La primera línea es un comentario, que el programa ignorará al ir precedido del punto y coma. La segunda es la definición de la acción, que nos dice que su número de identificación es el 0. Después tenemos el número de *collision boxes* de tipo 1. El nombre *Default* indica que éstos son validos para toda la acción (si pusiera *Clsn2: 2* sólo

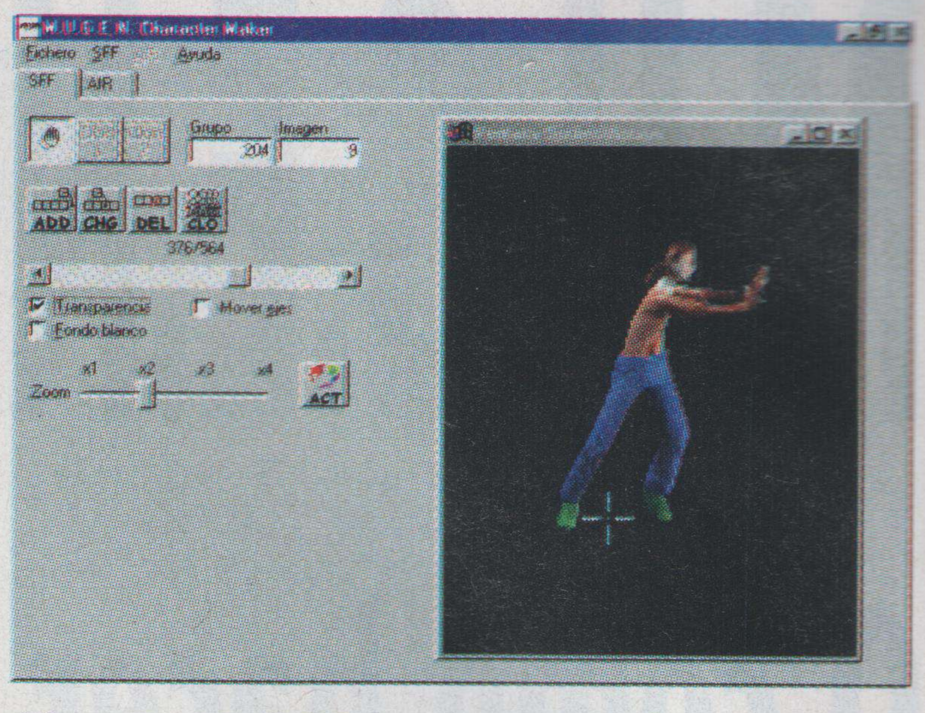


Fig. 2: M.C.M. es un excelente programa para crear personajes.

serían válidos para el primer *sprite*). Después están las coordenadas de cada *box*, que se corresponden con la imagen de la izquierda de la Fig. 3. Y finalmente tenemos la lista de sprites, cada uno con sus valores: el grupo y el número de sprite (descritos en el archivo *.sff*), sus coordenadas (normalmente 0,0) y el número de *frames* que durará el *sprite*. Tened en cuenta que el juego funciona a 60 *frames* por segundo, así que esta acción duraría dos segundos.

Otra cosa que debéis saber es que el programa tiene unas acciones obligatorias, que debéis poner siempre, y que describen las acciones básicas del luchador. Así, la acción 0 es la del luchador quieto, la 20 la de caminar, etc. Además hay unos *sprites* obligatorios en el fichero *.sff*. Podéis encontrar una lista completa de las acciones obligatorias en *mugen\docs\air.txt*, y de los *sprites* en *sprite.txt*.

Las animaciones del archivo *.air* nos permiten hacer muchas más cosas de las que hemos visto hasta

Cómo crear los sprites

Lo primero que uno necesita para crear un luchador son sus *sprites*, es decir todas las imágenes que describen al luchador en todos sus movimientos posibles. Hay muchas maneras de conseguir estos *sprites*; podéis crearlos desde cero con un programa de dibujo o con uno de modelado tridimensional, o también podéis tomar fotografías vuestras con una cámara digital. Pero la manera más fácil y popular es la de "robar" los *sprites* de un juego de lucha. A este proceso se le llama *ripping*.

Para hacer esta captura necesitáis mucha paciencia, pues se trata en esencia de hacer capturas de imagen del juego. Después deberemos limpiar de cada imagen todo lo que no sea el luchador, dejando sólo el color 0 de fondo.

Lo más normal para hacer esto es usar un emulador, pues hay muy pocos juegos de lucha para PC, mientras que la PlayStation o la Neo Geo tienen muchísimos. En cualquier caso, recordad que debéis tener los derechos de uso de los juegos (es decir, los juegos originales) para usarlos con el emulador.

Sin duda el mejor emulador para hacer esto es el NeorageX, que aparte de emular estupendamente la Neo Geo, tiene una genial herramienta para eliminar de la imagen todos los elementos que no pertenezcan al luchador. (Ver Fig. 7).

Direcciones de Interés

L*http://www.elecbyte.com*: Los creadores de M.U.G.E.N. Aquí encontraréis la última versión de este programa y de las *M.U.G.E.N. Tools*.

http://come.to/testp: Los testeadores oficiales han creado un juego completo usando M.U.G.E.N. Lo tenéis también en el CD de DIV Manía. Sus luchadores son los mejores para aprender el funcionamiento del programa.

http://go.to/mugen4ever: La página de fans más completa. Aquí están casi todos los luchadores que existen para M.U.G.E.N.

http://teleline.terra.es/personal/moi.ses: En la Página de Sés encontraréis el fantástico programa *M.U.G.E.N. Character Maker*. Muy recomendable si queréis crear un luchador.

http://perso.club-internet.fr/moah_/KOF91/: La competencia. Ésta es la página oficial de KOF 91 (Ver el cuadro correspondiente)

ahora, como por ejemplo invertir la imagen, hacer bucles o aplicar efectos de transparencia, una característica muy potente que podéis ver en la Fig. 4.

-luchador.cmd: Está dividido en dos partes. En la primera hay una lista de comandos, que no son más que los botones que hay que pulsar para hacer un ataque. Su descripción es similar a las acciones pero más sencilla. Veamos, por ejemplo, como se ejecutaría un *Hadoken*:

```
; Bola de Fuego
[Command]
name = Hadoken
command = ~D, DF, F, a
time= 30
```

Tras el comentario y la definición, viene el nombre del movimiento. Es importante que anotéis el nombre pues lo necesitaréis para activarlo después. Después viene la descripción del comando: soltar abajo (*~D*), pulsar abajo y delante (la diagonal), pulsar delante y a (un botón de puñetazo). Si queremos hacer un movimiento en el que haya que mantener pulsado un botón, por ejemplo atrás dos segundos, se haría así: *command= ~120$B, F, a* .Es decir, soltar atrás después de 120 *frames*, o 2 segundos, pulsar delante y a. Finalmente tenemos *time= 30*, que indica el tiempo que tenemos



Fig. 6: El luchador de la izquierda no tiene icono ni portrait.

para ejecutar la acción; si tardamos más no se activará.

La segunda parte del *.cmd* es más compleja y define cuándo podemos hacer cada movimiento para que, por ejemplo, no podamos hacer una patada aérea estando en tierra. Explicaré cómo funciona más tarde, pues su uso es muy similar al del siguiente archivo.

-luchador.cns: Describe los movimientos que el personaje puede realizar. Este es el último fichero que necesitamos, y sin duda alguna el más complicado. Llegar a dominarlo es casi imposible, especialmente porque la documentación referente a su uso está incompleta. Sin embargo, los movimientos básicos no son complicados de hacer.

El fichero *.cmd* es similar al *.air*, y así como éste se organiza en acciones, el *.cmd* es una lista de estados. Por ejemplo, al empezar el combate el luchador está en el estado de *Intro*, para después pasar al estado inicial; si pulsamos entonces arriba, el luchador pasará al estado de prepararse para saltar, y después al de saltar arriba, etc.

Por suerte para nosotros, la mayoría de estados ya están hechos, de manera que no tendremos que definir cómo caminar, saltar o agacharse. Los únicos estados importantes por definir son los ataques del luchador. Por eso veremos cómo se definen.

Un estado empieza por su definición, seguido de unos parámetros básicos. Después vienen los controladores. Los controladores son la parte más potente de todo el programa, y también la más complicada. Permiten al luchador hacer casi cualquier cosa, desde crear proyectiles a cambiar los colores del escena-



Fig. 5: Ejemplo de Helpers: Megaman y Galford con sus mascotas.

rio o crear un luchador ayudante (Ver Fig. 5). Para activar estas acciones se utilizan los *triggers*, o gatillos. Clarificaré esto con un ejemplo:

```
;Puñetazo Medio y
[Statedef 200]
type    = S
movetype= A
ctrl = 0
anim = 200

[State 200, 1]
type = HitDef
trigger1 = AnimElem = 6
attr = S, NA
damage    = 20
animtype = Hard

[State 200, 2]
type = ChangeState
trigger1 = AnimTime = 0
trigger2 = Time > 20
value = 0
ctrl = 1
```

Esto es más complicado que lo visto hasta ahora, y de hecho se puede complicar mucho más. Veámoslo por partes:

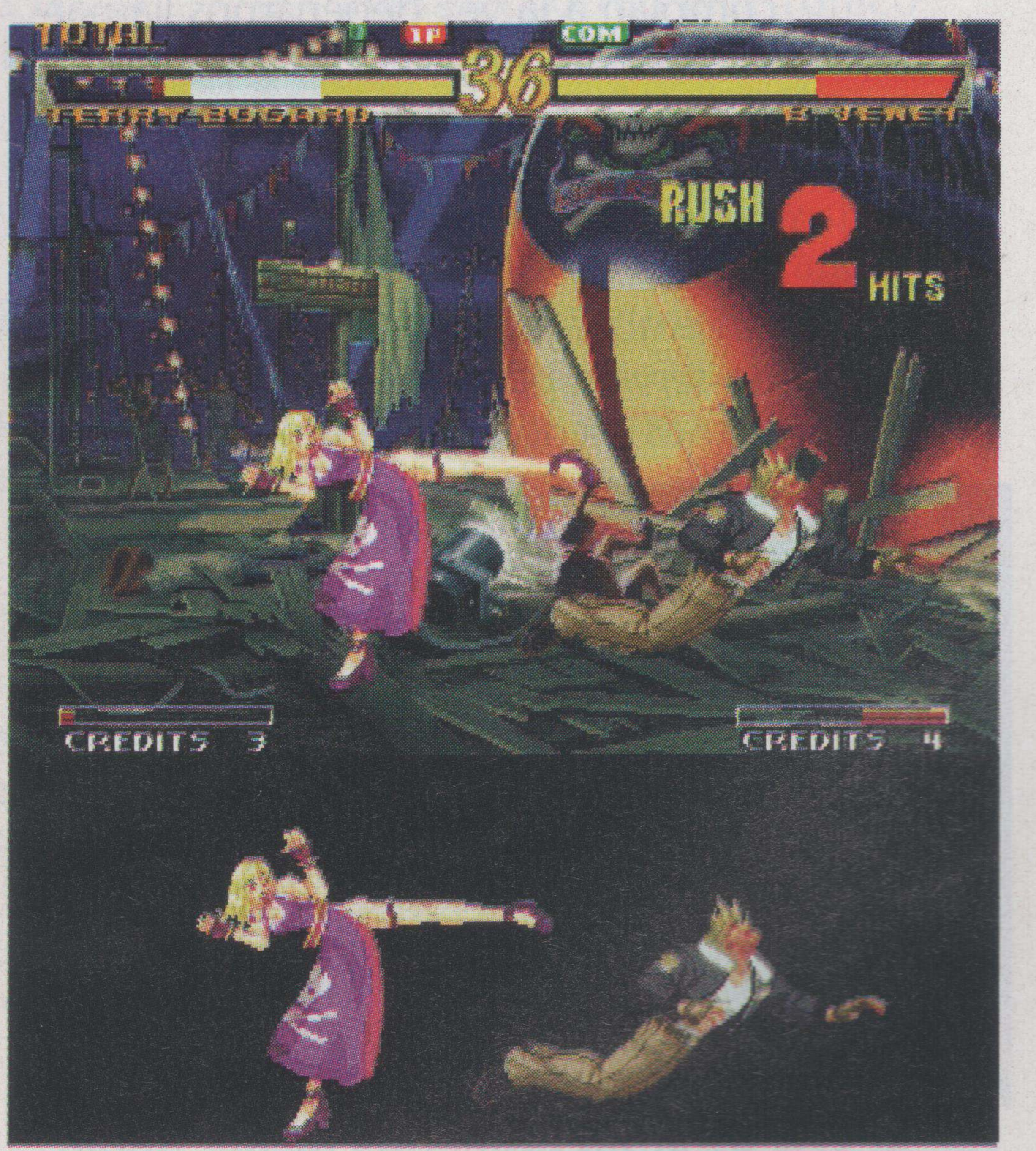


Fig. 7: Con NeoRageX el ripping de sprites es muy sencillo.

Primero viene el comentario y la definición. Después los parámetros básicos: tipo de estado o *type* (S- De pie, C- Agachado, A- En el aire, L- Tumbado en el suelo), tipo de movimiento (A- Ataque, D- Defensa, I- Todos los demás), si el jugador tiene o no control (0 para que no tenga control, 1 para que pueda interrumpir el estado y pasar a otro al pulsar los botones), y finalmente el número de animación. Para que el estado funcione correctamente deberemos haber definido esta animación en el *.air*, con sus *collision boxes* de ataque. Después viene un controlador (Cada estado puede tener tantos controladores como queramos). Tiene su definición y su tipo, en este caso definir un impacto o *HitDef*. Después viene un *trigger*, que es la condición que debe cumplirse para activarlo, en este caso que estemos en el sexto *sprite* de la animación. A continuación vienen los parámetros del estado. *Hitdef* es un estado muy complejo y puede tener muchos parámetros, por lo que os aconsejo que miréis detenidamente el ejemplo que viene en *player.cns*. En este caso tenemos los atributos (S- de pie, NA- Ataque normal), la energía que restará el impacto a nuestro contrincante y el tipo de animación que tendrá éste al recibirlo. (*Hard*, *Medium* o *Low*)

La flexibilidad del engine permite hacer supers con todo tipo de efectos

Finalmente tenemos otro controlador, que cambia al estado inicial y proporciona control al jugador cuando la animación llega a su fin o el tiempo es superior a 20. Para definir esta condición alternativa, se define un *trigger2*. Si, en cambio, quisiéramos que las dos condiciones se cumplan a la vez deberíamos llamarlo también *trigger1*. Para más información sobre los *triggers*, que sí están bien documentados, podéis consultar *mugen\docs\trigger.txt*.

Sin embargo, sólo con esto el estado no serviría para nada, porque no le hemos dicho al programa cuándo activarlo. Falta introducir el cambio a ese estado en la segunda parte del archivo *.cmd* (Atención: esto va en el archivo *.cmd*, no en el *.cns*):

```
;Puñetazo Medio y
[State -1]
type = ChangeState
value = 200
trigger1 = command = y
trigger1 = command != holddown
trigger1 = statetype = S
trigger1 = ctrl = 1
```

El estado -1 es un estado que siempre está activo sea cual sea el estado del luchador, y sus controladores sólo pueden ser de tipo *ChangeState*. Aquí vemos el que corresponde al golpe que acabamos de definir: el luchador pasará al estado 200 siempre que se cumplan simultáneamente cuatro condiciones: que hagamos el comando llamado "y", que no pulsemos abajo, que el luchador esté de pie y que tenga control.

Hasta aquí llega la descripción de los archivos. Pero ya hemos visto que el programa es complejo, y los archivos largos y numerosos. La pregunta que os debéis hacer es: ¿por dónde empiezo? Pues bien, aquí tenéis unos consejos para empezar.

Cómo empezar un luchador

Lo primero que necesitáis son los *sprites* del luchador (Ver el cuadro correspondiente). A continuación debéis descomprimir *player.zip* y renombrar los archivos con el nombre de vuestro luchador. Después deberéis modificar el *.def*. Seguidamente debéis crear el fichero *.sff* y el *.air* con las animaciones obligatorias, descritas en *mugen\docs\air.txt*. Esto es sencillo y gratificante, pues desde la animación 0 cada vez que queráis podéis cargar el programa y ver a vuestro luchador en acción.

Además, para que aparezca la imagen de vuestro luchador en la pantalla de selección de personajes tendréis que introducir su icono (imagen 9000,0 del *.sff*) y su *portrait* (9000,1). Si no lo hacéis al seleccionarlo su imagen estará en blanco, como veis en la fig. 6.

Y finalmente, debéis crear las animaciones de los ataques, sus entradas y comandos correspondientes en el *.cmd* y sus estados en el *.cns*. Y con esto ya tendríamos un luchador completamente funcional. Obviamente podemos hacerle muchas más cosas, como introducir sonidos, animaciones de introducción y final de *round*, ataques especiales, etc. Y por supuesto podéis crear su escenario con música, elementos animados, *scrolls*... Quizá podamos ver todo esto en un futuro artículo. Pero una vez entendidas las bases, todo lo demás llegará con la práctica. Espero que este artículo os haya servido de ayuda y ver pronto vuestros luchadores en la red.

Héctor Zapata
zapata@ncsa.es



Fig. 8: KOF 91 es un programa menos potente que M.U.G.E.N., pero más sencillo.

El otro M.U.G.E.N.: King of Fighters 91

En Internet existen otros programas que permiten personalizar en mayor o menor medida juegos de lucha. El más popular después de M.U.G.E.N. es un interesante juego para Windows llamado KOF 91. (Ver Fig. 8).

La filosofía de trabajo de este programa es muy distinta. Todos los luchadores tienen las mismas características y los mismos movimientos, y lo que los diferencia son los propios *sprites*. De manera que más que crear un luchador, lo que hacemos es substituir los *sprites* de un luchador existente por los nuestros, poniendo las imágenes *.pcx* en su directorio correspondiente.

Las ventajas son obvias: no hay complicados archivos de texto que modificar ni archivos de imagen y sonido que crear. Por lo tanto, crear un luchador puede ser cuestión de minutos una vez tenemos los *sprites*. Además posee un editor de luchadores muy fácil de usar. Pero ya os podéis imaginar los inconvenientes: no tenemos apenas flexibilidad, los luchadores tienen muchos menos movimientos y la jugabilidad en general es mucho peor. Pero también tiene interesantes añadidos, como por ejemplo un efecto de *zoom* muy conseguido. Además al ser para Windows es más fácil de usar.

Este programa está dirigido a aquellos que no tengáis ganas de complicaros la vida aprendiendo el funcionamiento de M.U.G.E.N., o para los que quieran hacer un luchador rápidamente. Pero probablemente os quedaréis con ganas de más, pues el programa es bastante limitado en cuanto a creación de luchadores.

MP3 y WINAMP

La revolución del audio en la red

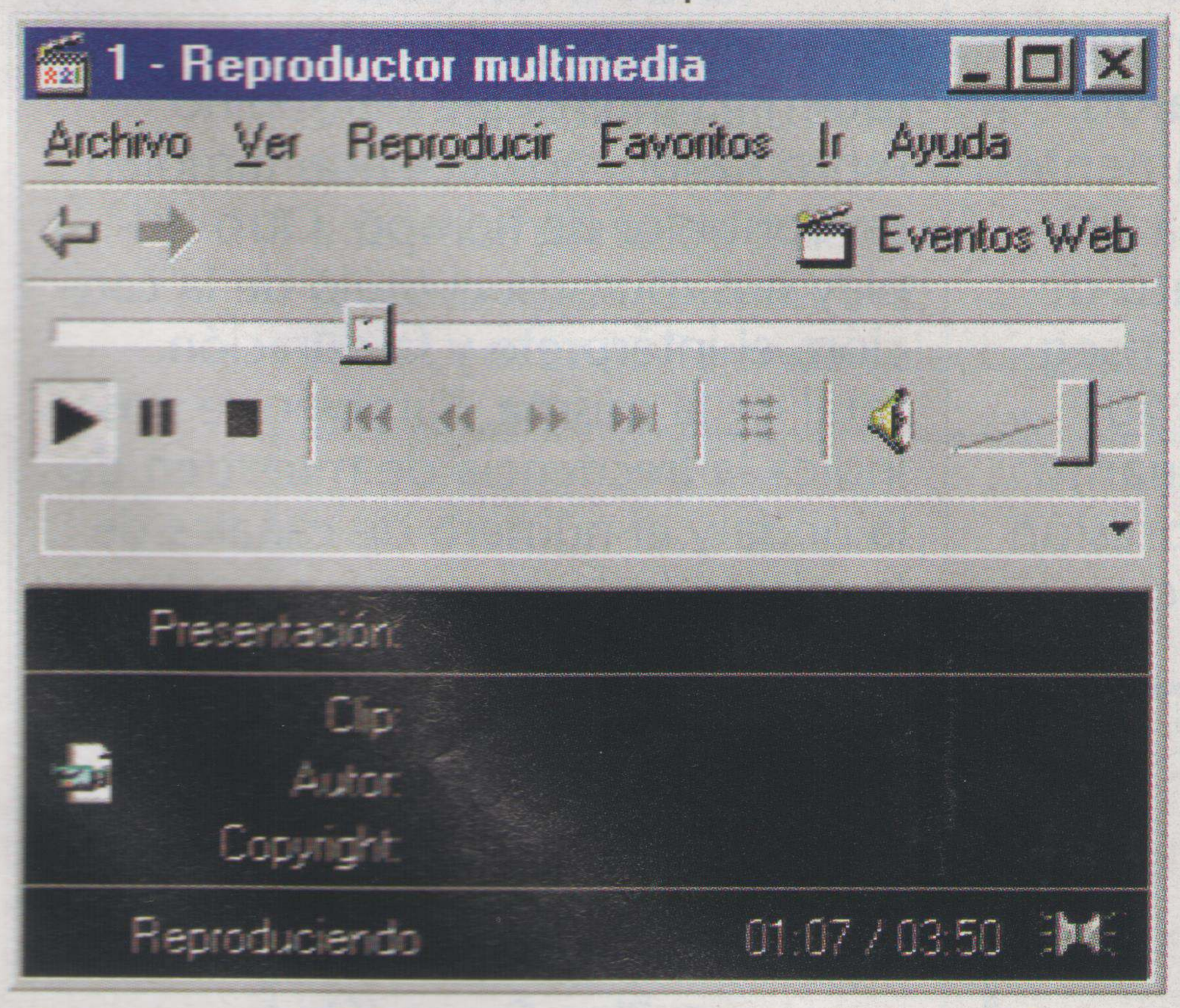


Veremos con detalle en este número de DIVmanía un formato sonoro que ha hecho cambiar radicalmente todo este submundo de la informática, abriendo un nuevo y amplio abanico de posibilidades. Nos referimos al flamante y casi mágico formato MP3.

El avance de la informática en general, se caracteriza por tener un crecimiento exponencial y todo lo relacionado con el sonido digital no se queda atrás: hace 10 años, un computador apenas era capaz de emitir algún que otro tímido pitido. Sin embargo, desde hace unos pocos años, los ordenadores son capaces de grabar, e incluso reproducir sonido extraído de fuentes externas, con una calidad idéntica a la original. El único inconveniente que se tenía hasta hace bien poco, era la gran capacidad de disco duro que se requería para almacenar sonido con una calidad aceptable, lo cual hacía imposible, inevitablemente, el intercambio de este tipo de archivos, tan enormes, a través de Internet.

Antes de nada, ¿qué es MP3?

MP3 no es ni más ni menos que un formato de compresión de audio.



El formato MP3 es un standard reconocido por Windows.

Antes de entrar en más detalles deberíamos intentar comprender lo que es un formato de compresión: pues bien, por compresión se designa a la acción de reducir de algún modo el volumen o espacio ocupado por un archivo. Podríamos dividir los formatos de compresión de ficheros en dos tipos:

- Los que comprimen uno o más archivos en un solo fichero, que únicamente podrá ser descifrado haciendo uso del programa descompresor correspondiente. Este tipo de formato de compresión es el que se utiliza para hacer backups, o para trasladar información en una cantidad arbitrariamente grande, de un ordenador a otro. Nunca podremos ejecutar o leer un archivo contenido dentro del fichero comprimido de esta manera, si antes no lo descomprimimos. Esta es la principal diferencia de este tipo de formatos de compresión con respecto a los del segundo tipo. Ejemplos de este tipo de formatos de compresión son *.ZIP, *.RAR, *.ARJ, *.AIN, *.ACE, etc.
- Los que permiten que el archivo comprimido sea leído y utilizado por programas concretos, realizando la descompresión en tiempo real, si es necesario. Es el caso de JPEG, MPEG o el que nos ocupa, MP3.

Las siglas MP3 representan en realidad la frase MPEG Audio Layer-3. Esto significa que MP3 es la parte sonora del formato de compresión de vídeo MPEG, que es el standard desarrollado por el MOVING PICTURE EXPERT GROUP para este efecto. Es por eso que, a partir de determinada versión de Windows 95, el propio sistema operativo provee de aplicaciones para reproducir archivos de vídeo en formato MPEG, y por consiguiente, también para reproducir archivos sonoros MP3.

¿Cuáles son las ventajas de MP3?

Como ya se ha podido vislumbrar en la introducción del artículo, la principal ventaja de MP3 es su poder de compresión: haciendo uso de un compresor MP3 seremos capaces de comprimir un fichero de tipo WAV de manera que pase a ocupar una décima, o incluso una doceava parte de lo que ocupaba originalmente, sin alterar perceptiblemente la calidad sonora del fichero original.

> La compresión MP3 se basa en la supresión de frecuencias no perceptibles por el oído humano.

Gracias a esto, ahora podemos transportar a través de Internet canciones completas, sin que esto suponga una espera eterna, y el consiguiente gasto telefónico. Veamos un ejemplo con cifras: Imaginemos que queremos enviar a través del correo electrónico una de nuestras canciones favoritas a un destinatario cualquiera. Pongamos que la duración de la misma es de 5 minutos y que queremos enviarla con calidad CD, es decir, sampleada a 44100 Hz, en estéreo, y con 16 bits por muestra. Realizamos las siguientes operaciones:

- Samplear con una frecuencia de 44100 Hz significa que se origi-



WinAmp es un reproductor de sonido que acepta MP3.



La interfaz de WinAmp.

narán 44100 valores cada segundo durante el proceso. Tenemos que multiplicar ese valor por dos, puesto que, al ser estéreo, tenemos dos canales por los cuales estamos recibiendo información.

- Debemos multiplicar de nuevo el valor obtenido por dos, ya que cada muestra, por ser de 16 bits, ocupara 2 bytes de memoria en nuestro disco duro.
- Por último, multiplicamos por 300 (segundos), ya que habíamos dicho que la canción duraba 5 minutos.
- Obtenemos por tanto, que la canción ocupa la friolera de 50 Mbytes.
- Si ahora multiplicamos los 50 Mbytes por 8 para obtener el número de bits de información que representan, y dividimos por los 28.800 baudios de un módem de calidad media, y por los 60 segundos que tiene un minuto, se obtiene que tardaríamos alrededor de 4 horas (245 minutos para ser exactos) para poder enviar este archivo a través de la red.

MP3 es un formato de compresión de audio muy eficaz y potente.

En caso de pasar a formato MP3 el WAV en discordia, nos hubiera bastado con unos 20 minutos, lo cual parece bastante más razonable.

¿Cómo funciona MP3?

Si hay algún lector que no haya escuchado todavía un archivo MP3, seguramente estará bastante escéptico, y le costará creer que no se pierda ninguna calidad de sonido si se logra un índice de reducción de espacio tan alto. Por otra parte, los lectores que hayan comprobado por sí mismos las cualidades de MP3, estarán pensando que es cosa de magia. Pues no se trata de ningún fenómeno paranormal. Es más bien una especie de truco, digno del mismísimo Copperfield, que consigue engañarnos con una eficacia sorprendente. La codificación MP3 se denomina perceptual, y esto viene a significar que se basa en el conocimiento de las características de nuestra percepción auditiva.

El proceso de compresión, que es bastante complejo, podría resumirse de la siguiente manera: se transforman las muestras o samples en valores de frecuencia aplicando una transformada rápida de Fourier (Fast Fourier Transform) y posteriormente se suprimen aquellas frecuencias que no son perceptibles por el oído humano, mediante la aplicación de un modelo psicoacústico (psicoacustic model). Es por eso que no notamos diferencia al reproducir los archivos MP3, cuando en realidad ha sido modificada considerablemente la onda original.

Limitaciones de MP3

Aunque MP3 es un potentísimo formato de compresión, tiene un inconveniente: como la descodificación o descompresión se realiza en tiempo real, esto toma un tiempo, lo cual origina, inevitablemente, un pequeño retardo. Este retardo es aproximadamente de 50 mseg en el caso de la primera versión de este formato de compresión (Layer 1), 100 mseg en la segunda (Layer 2) y 150 mseg si se trata de MP3. Cuanto mayor es el índice de compresión mayor es el retardo. Esto no representa un problema para el transporte masivo de sonido por Internet (los retardos citados son despreciables), pero esto puede suponer un problema para el uso de MP3 en el campo de la telefonía, ya que el retardo acumulado puede llegar a hacer las comunicaciones bastante torpes. Para aplicaciones telefónicas es preferible reducir directamente la frecuencia de muestreo, reduciéndose así la calidad del sonido hasta el límite en el que las palabras siguen siendo comprensibles (formato PCM).

Otra limitación de MP3 es que sus reproductores requieren de cierta potencia de CPU, ya que realizan, como ya hemos dicho, descompresión en tiempo real. Sería imposible reproducir MP3 con plena calidad en un 486 DX2, por ejemplo.

Programas que pueden reproducir MP3: WinAmp

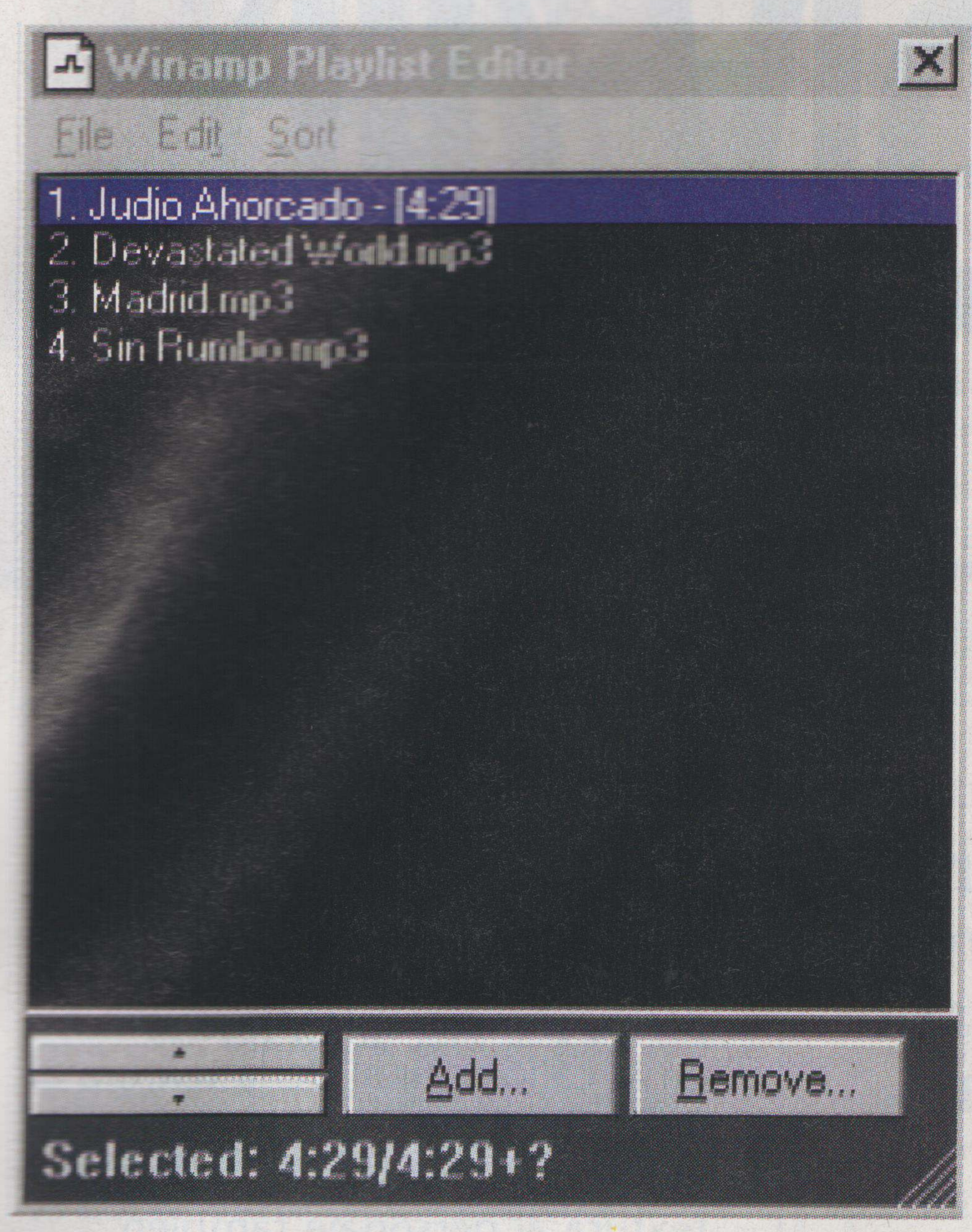
El reproductor multimedia de Microsoft puede reproducir MP3, pero no nos ofrece la posibilidad de modificar las características del sonido, ni de manejar un grupo grande de ficheros. Todo esto y más lo encontramos en el que es, posiblemente, el programa de reproducción de MP3 más extendido. Nos referimos a WinAmp.

El interfaz de WinAmp

WinAmp es un programa reproductor de numerosos formatos de audio entre los que se encuentra MP3. Su interfaz emula al frontal de un típico aparato reproductor de Compact Disc, permitiendo cambiar la apariencia del mismo con las diferentes pieles o "skins" que podremos adquirir en cualquiera de los sitios web relacionados. La interfaz tiene un display que representa, en forma de barras volumétricas o bien en forma de onda (esto se cambia haciendo clic en el display con el ratón), la intensidad del sonido reproducido, apareciendo también un contador de tiempo. Otro display nos muestra el título de la canción, el intérprete y la duración total de la misma. También se nos indica la frecuencia de muestreo de la onda y el número de canales de la misma (estéreo o mono). Podremos variar el volumen y la panorámica con dos controles de desplazamiento horizontal. Así mismo se dispone de todos los controles típicos de manejo de un reproductor de CD convencional, como son Play, Pause, Stop, FastForward, Rewind, Canción siguiente, Canción anterior,

Rendimiento de cada una de las capas sonoras MPEG

Capa	Factor de compresión	Retardo
Layer 1	1:4	< 50 mseg.
Layer 2	1:6 - 1:8	100 mseg.



WinAmp permite crear listas ilimitadas de reproducción.



El ecualizador gráfico de WinAmp.

Introducir nuevos archivos (es decir, cargarlos desde disco) e incluso los usuales Shuffle (Random) y Repeat.

Todos estos elementos pueden variar ligeramente dependiendo del skin que estemos utilizando, pero básicamente, dispondremos de los mismos controles.

Las opciones de WinAmp

Haciendo "clic" en la parte superior izquierda de la ventana de WinAmp se nos despliega un menú con diferentes opciones.

> WinAmp es el reproductor de formato MP3 más generalmente extendido.

Son las siguientes:

- WinAmp: muestra información sobre el programa en general, como los créditos de realización del mismo, el precio que hay que pagar para registrarlo (WinAmp es Shareware, claro), las mejoras de la versión en concreto o direcciones URL (web) de páginas relacionadas con el tema.
- Play File: Permite seleccionar uno o varios ficheros ubicados en nuestro disco duro, que pasarán a formar parte de la lista de reproducción de WinAmp.
- Play Location: Permite seleccionar un fichero ubicado en una máquina remota, a través de Internet, para reproducirlo desde nuestro ordenador.
- View File Info: Nos da información sobre el fichero que actualmente está siendo reproducido (título de la canción, autor, álbum, año de creación, etc).
- Graphical Equalizer: Nos da acceso a un ecualizador gráfico de 10 bandas que oscilan entre 60 Hz y 16 Khz, así como a un control de preamplificación.
- PlayList Editor: Lista de archivos a ser reproducidos. Se nos da la posibilidad de ordenarlos por título, por nombre de fichero o aleatoriamente.
- Options: Esta opción despliega otro menú que nos da acceso, entre otras cosas, a las preferencias del programa y al selector de skins (pieles). Podremos encontrar pieles para todos los gustos: agresivas, clásicas, sobrias, coloristas e incluso temáticas (desde Star Wars a Los Simpsons, pasando por Spiderman). También podremos cambiar otros parámetros como el tamaño de la ventana de WinAmp.
- PlayBack: Estas opciones son accesibles directamente desde la interfaz.
- Visualization: Permite modificar el plug-in preseleccionado. Este plug-in podrá ser un volúmetro u osciloscopio determinado de los que tengamos disponibles en nuestro WinAmp, que será mostrado a pantalla completa. Algunos de ellos son realmente atractivos y podremos activarlos pulsando con el botón derecho del ratón sobre uno de los displays de la interfaz.

Dónde encontrar todo este material

Para encontrar todo tipo de archivos MP3 podremos dirigirnos a páginas como www.mp3.com que nos da acceso a gran cantidad de canciones, escrupulosamente clasificadas por estilos. Podremos tanto bajarlas de la red (download), como escucharlas directamente. En el caso de bajarlas podremos optar por diferentes calidades, para minimizar el tiempo de download, si así lo deseamos. De todas formas es conveniente hacer uso de alguno de los programas destinados a acelerar la transferencia de archivos en Internet, como Netants, ya que, aunque MP3 ahorra mucho tiempo y espacio, los archivos siguen siendo bastante extensos (una media de 4 a 5 Mbytes).

Para buscar música independiente o autoproducida podemos buscar en www.vitaminic.com, que ofrece la posibilidad de dar publicidad y comercializar sus obras en formato MP3, a los músicos tanto profesionales como amateur. Para conseguir la última versión de WinAmp bastará con navegar hasta www.winamp.com.

Sergio Cánovas

Formatos que es capaz de reproducir Winamp

Formato	Extensión
Impulse Tracker Module	*.IT
Extended Module	*.XM
ScreamTracker 3 Module	*.S3M
ScreamTracker 2 Module	*.STM
MultiTracker Module	*.MTM
ProTracker Module	*.MOD
Layer 2 MPEG	*.MP2
Layer 3 MPEG	*.MP3



Posando para DIV (III)

Aplicando texturas

En el artículo anterior estudiamos cómo crear distintas posturas secuenciales que nos diesen la sensación de movimiento. Hoy veremos un punto que ha provocado el mayor número de consultas por vuestra parte: la aplicación de texturas.

Dado el interés que habéis demostrado, en este número comentaremos los métodos para añadir realismo y color a nuestros personajes Poser, aplicando imágenes bitmap a su superficie.

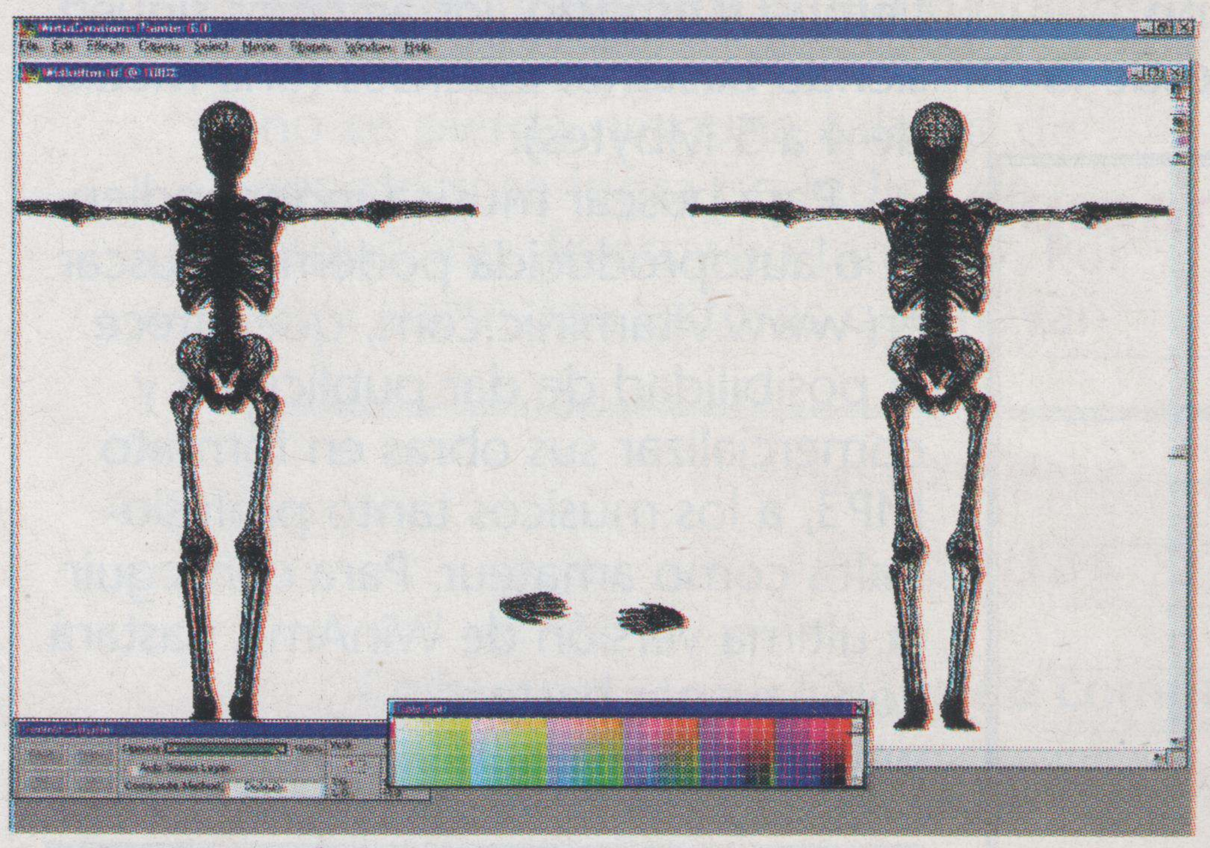
Quizás es el primer paso a la hora de diseñar vuestros personajes del juego, que seguro os traéis entre manos: definir el "aspecto" que tendrá el personaje para luego aplicar el movimiento y animación y posteriormente, crear los sprites correspondientes.

Generalmente, al crear el guión de los juegos, se suele discutir el aspecto que tendrá el personaje: "Detective joven con camiseta blanca y pantalones vaqueros", "Elfo de piel blanca, con chaleco verde y botas rojas", etc.

Pues eso es precisamente lo que haremos a continuación, definir el aspecto de nuestro personaje.

Aplicando texturas

Llegamos al momento que tantas consultas ha provocado en mi buzón: el personalizar nuestro personaje aplicándole los colores y ropas que deseemos. Para ello, como siempre, debemos tener muy claro antes de empezar, las características de nuestro personaje, en definitiva, "crearle una historia": ¿ha luchado cuerpo a cuerpo y tiene cicatrices? ¿su ropa es recién comprada o está llena de barro y polvo? ¿es de piel blanca o está tostado por el sol?



La textura del esqueleto.

También debemos decidir cómo vamos a crear la textura. Vaya esto por delante: cualquier programa de dibujo sirve para realizar las "pieles" de nuestros personajes en Poser. Es decir, cada uno puede utilizar su programa de retoque favorito: Photoshop, Paint Shop, Painter, Paintbrush (que viene por defecto con Windows), e incluso el propio editor de Div.

Obviamente, existen otras opciones más "profesionales" que seguro que todos aquellos que manejáis programas 3D conocéis: me refiero a las aplicaciones de texturizado como Deep Paint 3D o la propia de Metacreations 3D Paint. Estas utilidades te permiten pintar directamente sobre un objeto 3D, como si se estuviera pintando una maqueta o un lienzo y nos muestran en todo momento el resultado final de nuestras pinceladas. Así, sólo habría que cargar en el programa de texturizado el personaje de Poser (guardado antes desde este programa en alguno de los formatos 3D que reconozca vuestro texturizador: 3DS, OBJ, etc...) y aplicarle la textura. La imágen bitmap generada por esa textura debe guardarse después en alguno de los formatos reconocidos por Poser (por ejemplo, BMP).

Para los que utilicéis este método de texturizado (programas *ad-hoc)*, os doy una serie de consejos que seguro os son de utilidad:

- Haced que el tamaño del bitmap de la textura sea lo suficientemente grande (1024x1024 pixels, por ejemplo), pues influye en la calidad final de la textura al aplicarla.
- El método de aplicación de la textura al personaje también influye. Si desde un programa 3D escogemos un mapeado cilíndrico, cúbico, plano, etc... para aplicar al objeto Poser antes de texturizar, esto tendrá una influencia decisiva en el resultado final.
- Moved la luz en el programa de texturizado, pues una zona en sombra que creamos que está bien pintada, puede presentar pinceladas mal dadas, trazos sin cubrir, etc...

De todas formas, sabiendo que el presupuesto de casi todos es muy reducido como para permitirnos adquirir una de estas utilidades de aplicación directa de texturas, lo haremos con nuestro programa de retoque gráfico tradicional. Yo utilizaré Painter 6 de Metacreations, porque es el que me ha parecido que se adapta mejor a mi forma de trabajo. Vosotros podéis usar cualquier programa, lo único que se requiere es que sea capaz de trabajar con partes de imagen cortadas y pegadas (el editor de Div y el Paintbrush que viene de serie con Windows pueden hacerlo).

Como ejemplo y dado que cuando estoy escribiendo el artículo quedan pocas fechas para la Eurocopa de fútbol, haremos un pequeño homenaje a la Selección



Los elementos iniciales.



Pongámosle cara a nuestros personajes.

Española creando un personaje que será un jugador de la misma, para un juego de fútbol.

Los datos de partida son sencillos: en cualquier revista, periódico o página Web encontraremos fotos del uniforme de la Selección, así que sólo tenemos que copiar el diseño de la camiseta y pantalón nacional para que nuestro personaje cobre vida. En concreto, yo seleccioné de Internet la imagen que podéis ver a la derecha. De ahí puedo copiar el dibujo de la camiseta y por otra parte, aparece el escudo de la Selección bien claro. Todo esto nos será de gran utilidad en los siguientes pasos.

Por otro lado, para darle un aire más personal, vamos a ponerle al jugador la cara que nos parezca mejor. Si tenéis cámara digital, sólo tenéis que fotografiar a la persona que queráis y pasar la foto al ordenador como JPG. También se puede seleccionar una foto que tengamos y escanearla. El caso es tener una imagen JPG de la cual podamos "cortar y pegar" una cara para añadirla a la textura de nuestro jugador. En los próximos minutos veremos cómo hacerlo.

Las plantillas de texturas de Poser

Cada modelo de persona de Poser (hombre de negocios, niño desnudo, etc...) tiene una forma distinta definida por las arrugas de la vestimenta, calzado, etc... De ahí que la forma de pintar su textura es diferente.

Metacreations nos facilita las cosas dándonos las texturas básicas de cada figura disponible en Poser. Si abrimos el directorio "Templates" en el CD de Poser, veréis que hay una serie de dibujos en formato TIF, que pueden ser abiertos en cualquier programa de dibujo. Abrid por ejemplo, "Mskelton.tif" y podréis observar que lo que ahí se refleja es el dibujo de los polígonos que componen el modelo de esqueleto masculino de Poser, dividido por partes y en la postura básica (con los brazos extendidos). Las manos se encuentran separadas de la figura central y el cuerpo se muestra de frente y de espaldas. Lo que representa la plantilla es un "desenrolle" (un "unwrap" en inglés) del objeto Poser en todos los polígonos que lo componen. Si pintamos directamente sobre esta imagen y la cargamos posteriormente en Poser, adquiere la pintura que hayamos añadido a esta imagen bitmap. Puede sonar complicado, pero ahora veremos que no es así.

Así, cada modelo de Poser lleva su textura correspondiente, sólo hay que mirar el nombre en inglés de cada una para imaginar su correspondencia con los modelos del programa.

Dado que Poser no incluye ningún modelo en pantalón corto y camiseta para nuestro personaje, lo que haremos será pintar la textura de "Hombre vestido informalmente" (Casual man), simulando la camiseta y el pantalón corto de la Selección.

Comencemos por abrir en nuestro programa de dibujo la plantilla "CASMAN.TIF" del directorio "Templates" del Cd Poser. Como podéis comprobar, se trata del modelo de hombre informal dividido en polígonos, se puede reconocer dónde acaban las mangas de la camiseta, la cintura, etc...

Lo normal es comenzar pintando las partes que quedan al descubierto, como los brazos, piernas, cara, etc..., seguir con la vestimenta y acabar con el calzado. Así que procederemos a ponerle cara a nuestro personaje y pintaremos el resto de cuerpo descubierto.

Recortad de la imagen JPG que deseéis la cara de la persona a "retratar" en el personaje. Todos los programas de dibujo tienen una herramienta para seleccionar zonas en el dibujo, así que seleccionad la cara de la foto que os guste y "cortad" la selección (en casi todos los programas suele estar bajo "Edición>Cortar"). Ahora tenéis la cara de vuestra imagen en el portapapeles de Windows y sólo hay que pegarla en la plantilla que hemos abierto CASMAN.TIF. Como podeis ver arriba yo he tenido que disminuir el tamaño de la cara recortada y rotarla un poco para que quede recta.

Pegad la cara y, si como me ha sucedido a mí, no encaja con la de la plantilla por ser muy grande, no estar completamente recta, etc... rotadla y escalarla con vuestro programa de dibujo, hasta dejarla con las dimensiones de la plantilla de nuestro personaje. Una vez encajada la cara, seleccionad el color de la piel de la cara pegada (generalmente es la herramienta "cuentagotas" en todos los programas de dibujo, la que permite seleccionar un color de los que aparecen en una imagen). Ya tenéis el color de la piel de vuestro personaje tomado de la realidad. Con la herramienta de pincel, pintad las zonas correspondientes al resto de la cabeza, cuello y brazos (las piernas las dejamos para luego) tanto de frente como de espaldas. No os preocupéis por seguir los contornos de la plantilla, en la figura Poser lo único que cuentan son los polígonos que aparecen en la plantilla, así que todo lo que pintéis que sobresalga no aparece en el personaje al colocarle la textura.



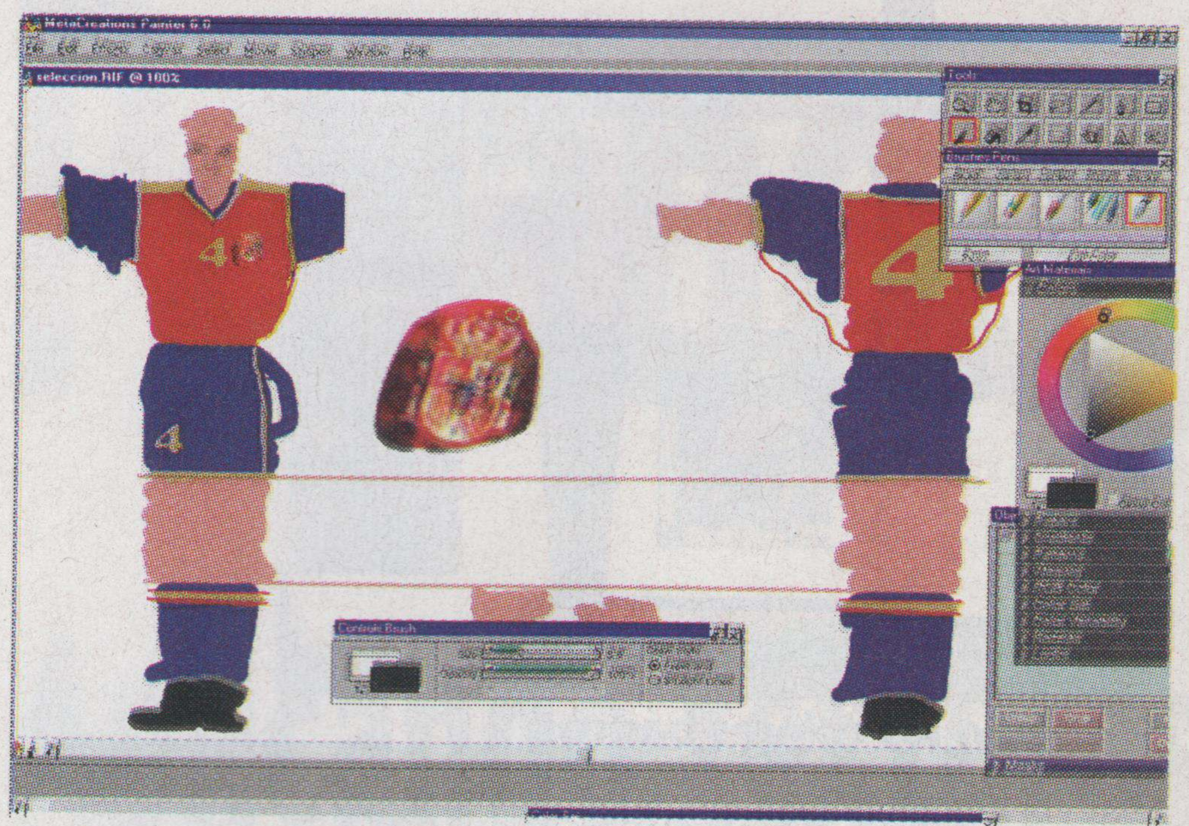
Pintando los colores básicos.

Una vez que tenemos la cara, comenzamos con la camiseta. La Selección Española lleva un escudo bastante grande sobre el pecho, como dibujarlo a mano puede quitar realismo a nuestra imagen, vamos a hacer igual que con la cara del personaje: recortarlo y pegarlo. De la imagen que tengamos de jugadores de la Selección, recortamos el escudo de esa imagen y lo pegamos en nuestra plantilla. Puede suceder, al igual que ocurrió con la cara, que haya que disminuir su tamaño y rotarlo para que quede recto. El resultado final podéis verlo en la página siguiente.

> Se puede dar a la figura un toque personal añadiendoles una cara

Ahora sólo queda seleccionar el color de la camiseta nacional (de nuevo la herramienta "cuentagotas" de vuestro programa de dibujo viene en vuestra ayuda). Coged el color de la camiseta en la foto de la que hayáis recortado el escudo y comenzad a pintar nuestra plantilla Poser de rojo.

Como en la foto que he seleccionado previamente no se apreciaban todos los detalles de las mangas y pantalones, descargué otra imagen de la Selección en la que sí se veían todas las partes del uniforme. Una vez observada la foto, y siguiendo el procedimiento de tomar el color de la foto origi-



La textura completa.

nal ("cuentagotas") y utilizarlo para pintar las líneas de la camiseta de la textura, añadimos las mangas azules y los detalles amarillos del cuello y mangas. El resultado debería ser como el de abajo.

Ya podemos poner el número a la camiseta. Con la ayuda de la herramienta de texto de vuestro programa de dibujo, crear un número en la espalda y en el pecho de la textura.

Creemos pantalones cortos de la nada

Dado que la plantilla CASMAN.TIF corresponde a un personaje con camisa de manga corta y pantalones largos, debemos simular que los pantalones son de deporte. La nueva versión de Poser ya permite modificar las partes de la ropa para que incluyamos las que más se ajusten a nuestro personaje, pero con Poser 2 debemos seguir el procedimiento que os comento a continuación.

Podemos tomar una muestra del color de la piel de la cara para el cuerpo

En primer lugar. marcad el espacio que existe entre el final del pantalón corto y el comienzo de las medias del jugador. Esto es, cread dos líneas rectas horizontales que marquen el espacio que corresponde a la pierna descubierta. Dichas líneas deben extenderse desde la parte correspondiente a la figura de frente, hasta la figura de espaldas, para que después coincidan.

Posteriormente, sólo queda rellenar con el mismo color que la nueva zona que hemos creado para las piernas.

Por último, coloread del mismo modo que hemos hecho con la camiseta, el pantalón, medias y calzado del jugador. Tened en cuenta que el resultado de nuestro trabajo es el básico para crear sprites posteriores, añadiendo luces, animación y otros objetos en Poser. Tal y como hemos concluido, el trabajo quedaría estupendamente para sprites de tamaño medio, pero si queremos hacer una "intro" en formato película, con algún plano cercano, etc...deberíamos añadir nosotros mismos luces y sombras a las arrugas de la camiseta, barro a las botas, sombras a la cara y cuello, simular pelo en los brazos, etc...pero eso depende ya del sentido artístico de cada uno.

Aplicando la textura en Poser

Nuestra textura de la Selección debemos guardarla en formato BMP o bien TIF, dado que son los únicos formatos gráficos que admite Poser 2 para cargar texturas en los modelos.

Una vez la tenemos guardada con el nombre que deseemos, sólo tenemos que abrir Poser, y desde el menu "Render" seleccionar "Surface Options". Se nos abrirá una ventana en la cual podremos seleccionar en un desplegable la figura a la cual vamos a aplicar la textura ("Figure 1", etc...), y cargar las texturas correspondientes a "Bump" (ver la sección siguiente) y a "Texture Map" (la que hemos creado).

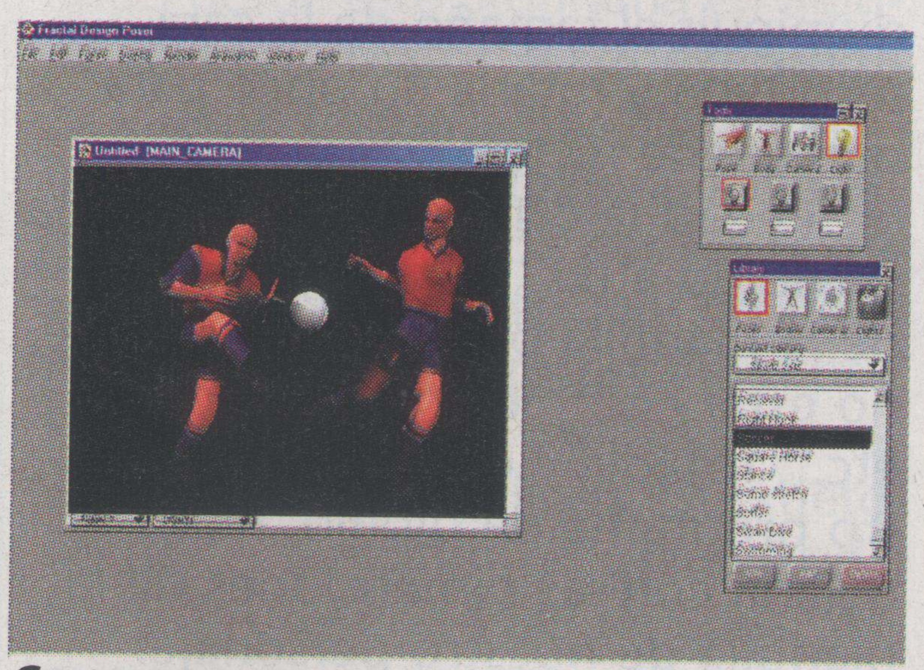
Pulsamos el botón "Load" en "Texture Map" y seleccionamos la imagen TIF o BMP de la textura que hemos guardado. A partir de ahora podremos modificar las posturas, crear animaciones, etc... con nuestro personaje, que la textura corresponderá a la que hemos indicado a Poser.

Probad a hacer un Render de vuestro personaje, y comprobaréis los resultados de vuestro trabajo previo. Con éste método podréis crear los sprites de vuestros personajes en la mitad de tiempo que en el editor de Div "a mano", con resultados infinitamente mejores.

Tipos de texturas: Map y Bump

Como comentábamos, en el cuadro de carga de texturas de Div se nos da la opción de seleccionar dos tipos de texturas diferentes. Veamos en qué consiste cada una de ellas.

Bump: La traducción literal es "abolladura", pero este tipo de mapas lo que nos indican es el relieve de la textura. No contienen color, son en blanco y negro y, según se pinte una zona más clara u oscura, nos indicará las partes más resaltadas del modelo.



Creamos una escena con nuestra textura.

Sonará lioso, pero en realidad es bastante fácil. Abrid por ejemplo, la textura "Male Muscle Texture.TIF", que corresponde con la textura Bump del hombre desnudo. Veréis una imagen en blanco y negro que simula la musculatura de un hombre.

En esta imagen, las zonas blancas son las zonas que "sobresalen" de la figura. Por ejemplo, los abdominales son casi blancos puesto que son músculos que dan relieve a la figura.

Por contra, las zonas más oscuras (gris oscuro y negro) indican "hundimientos" en la superficie de la figura. Así, la zona donde están los abdominales es más oscura que el resto, para indicar que está a un nivel inferior.

Si quisiéramos crear una textura Bump para nuestro uniforme de la Selección, construiríamos una superficie gris medio (indicaría las zonas planas), sobre las que aplicaríamos gris claro y blanco en el punto más alto de las arrugas, y gris oscuro y negro en los dobleces.

Este tipo de textura da mucho realismo a la ropa, sobre todo a los músculos de las figuras con cuerpo descubierto, pero consume tiempo de render y debemos evaluar si son necesarios para nuestra figura. Si vamos a crear sprites de 100x100 pixels es inútil realizar mapas de relieve, pues no se apreciarán. Si queremos realizar una animación con planos cortos y medios de los personajes, probablemente sea necesario crear un mapa Bump para ellos.

Los Map Textures son las texturas que hemos creado a lo largo del artículo. Definen los colores que tienen los personajes, así como los detalles de la vestimenta, etc... Una textura de color bien trabajada es la que marca la diferencia entre un personaje mediocre y uno bueno a la hora del render.

Y esto es todo por hoy. Próximamente veremos cómo diseñar la librería de animación con las posturas creadas en el capítulo anterior, y crearemos los sprites correspondientes para cargarlos en Div.

Como siempre, recordaros que podéis realizar las consultas que deseéis en el e-mail:

tayete@teleline.es.

Santiago García Mazariegos "Tayete"

El correo del lector

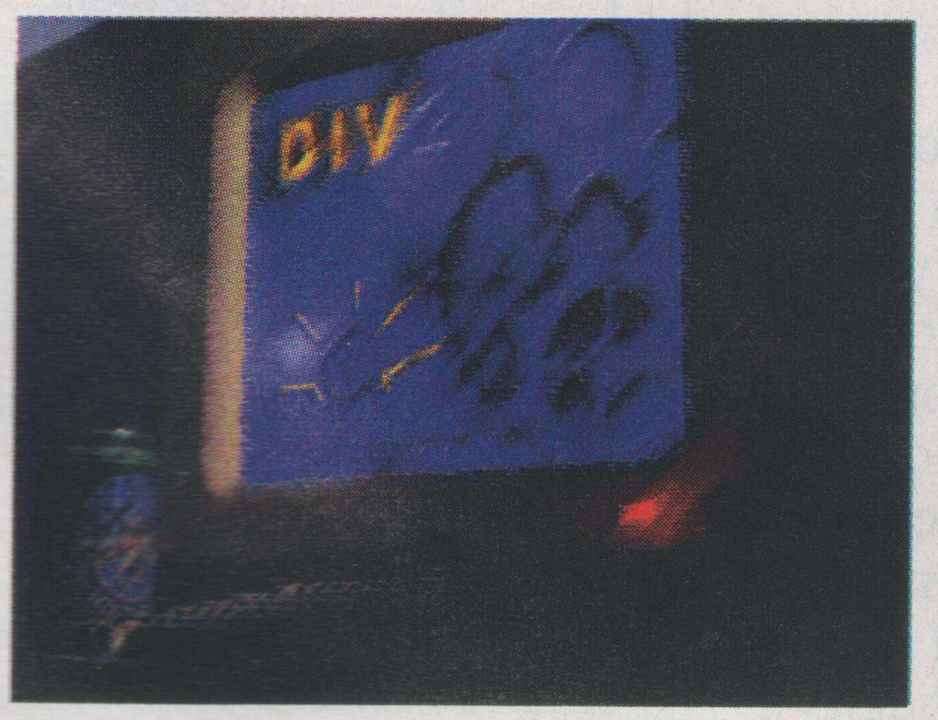


Para vosotros

Un número más, aquí tenéis vuestro espacio para salir a la luz pública. Para poneros en contacto con otros diveros, para intercambiar ideas o, para criticarnos o alabarnos y, en general, para decir lo que os apetezca, siempre y cuando tenga que ver con DIV claro está. No nos cansamos de repetir que esta revista es tanto vuestra como nuestra.

Crítica constructiva

Hola, quería comentar algunas cosas que me gustan y otras que no de la revista DivMania:
- El tema del concurso para enviar los juegos está muy bien, pero se limita solo a Div, somos un monton de personas que leemos la revista y programamos en otros lenguajes como C, Paskal, etc... ¿Por qué nosotros no podemos participar? Me parece una manera de limitar la revista a solo Div.
- Me gustan sobre todo las secciones que tratan sobre programacion con DirectX, Utilidades con MFCs, OpenGl, etc. De esta manera no se limita sólo a Div. En el fondo la gente que se inicia programa en Div, pero llega un momento que con Div no es suficiente y aprenden a programar en C bajo Windows. Así que estas secciones son de bastante utilidad.
- Las fotos que ponéis junto al texto me parecen demasiado exageradas, me explico: igual se está comentando cómo programar la parte de hablar con otros personajes y al lado en vez de salir una foto del ejemplo de la revista sale una foto del King Quest VIII o del Baldurs Gate . ¿¿?? Y todos sabemos que el código que se esta tratando no le llega a la suela de los zapatos de esos juegos.
Bueno, en resumen que la revista está muy bien que espero que sigáis así pero ke no os limitéis tanto a DIV o en caso contrario que penséis en crear una revista nueva *DirectXMania* o *Cmania*. Un saludo. *JVMA*



Gracias por tu carta. Tomamos nota de tus sugerencias. Respecto a la participación de juegos realizados en otros programas, sintiéndolo mucho no es posible. Sin embargo estamos abiertos a publicar cualquier juego que nos enviéis y que os haga ilusión ver publicado. Sobre la limitación a Div Games Studio, querido amigo, Divmanía está hecha por y para el Divmaniaco y aunque toquemos otros temas siempre estarán relacionados con Div. Si no lo hiciésemos así perderíamos nuestra razón de ser. Lo de hacer un DirectXManía... pues es una idea.

Dudas

Hola, soy Victor Arago, un lector de Divmanía. En el número 8 explicáis cómo hacer mapas de durezas pero no me he aclarado. Podríais ayudarme. Espero vuestra contestación.

Lo mejor es que te pongas en contacto con el autor del artículo. De cualquier modo deberías concretarnos qué es lo que no entiendes porque de otro modo no podemos ayudarte.

Mensaje de la redacción

Un mensaje de los que hacemos esta revista para Alberto Zurriaga. Para resolver tus dudas lo mejor sería que te pusieras en contacto con Hammer Technologies, los creadores de DIV Games Studio que son quienes mejor pueden informarte. Nosotros sí que podemos contestarte a dos cosas: una que lo mejor para aclararte es que te compres el DIV original (el pirateo es un delito) y que su precio es muy asequible teniendo en cuenta que cualquier software de programación se sube por encima de las 40.000 pts.

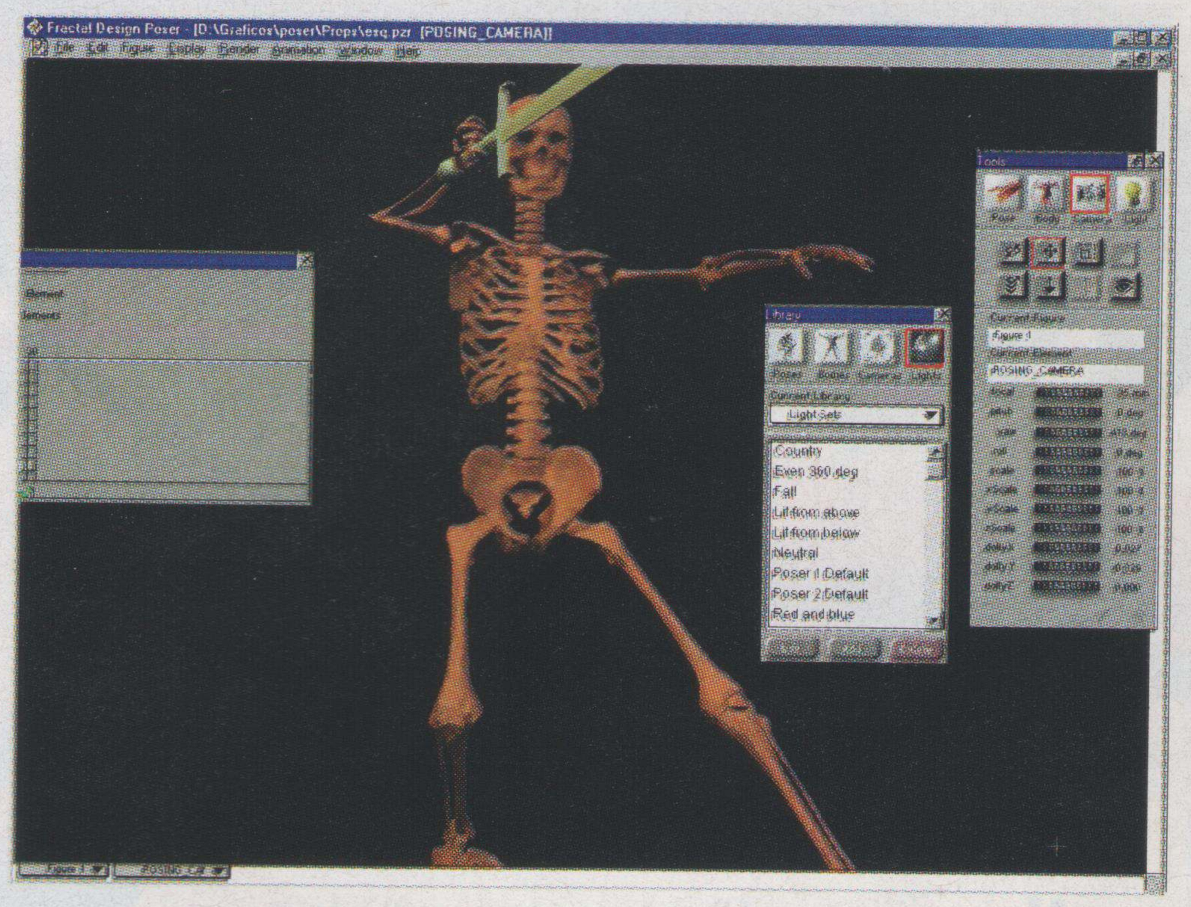
Corrector ortográfico

Hola. En primer lugar quiero felicitarles por la revista DIV manía. Para gente como yo que acabamos de emprender el viaje alucinante dentro del mundillo de la programación es una fuente inagotable de trucos e ideas.
En el CD de este mes había una corrector ortográfico. Antes de instalar la aplicación no sospechaba que iba a hacer tantas cosas. La corrección de cualquier programa en tiempo real me parecía una idea genial y la calculadora era exactamente lo que necesitaba. Sin embargo, hay un montón de funciones que no están muy bien explicadas en la ayuda. ¿Qué significa toda la información sobre la jerarquía de las ventanas? ¿Cómo puedo corregir el código fuente de C++? ¿Cómo puedo creer un diccionario nuevo con los nombres de mis amigos sin añadirlas a los diccionarios principales? *Pablo Olabarrieta.*

Está muy claro, amigo mío. Lo que necesitas es ponerte en contacto con su creador:
Anthony Rostron, ajr@arrakis.es.

Sugerencias

Hola, soy un nuevo usuario de Div2 y por casualidad vi la MAGNIFICA revista DIVmania en el kiosko, sin dudarlo un instante me la compré. Mi problema es que como no tengo los anteriores números no puedo seguir los cursos como el de la aventura grafica, RPG, etc. (menos mal que he encontrado esta página. Para mejorar esto, podrían poner en el mismo apartado de las suscripciones un mensaje(ahora no sé como lla-



marlo) para que te envíen números atrasados de DIVmanía. Gracias por vuestra atención, se despide Jordi.

Parece que todavía hay despistados. En fin, volveremos a repetir que para conseguir números atrasados debéis poneros en contacto con el departamente de suscripciones en el número 91 304 06 22 o bien en la dirección suscripciones@prensatecnica.com. Tomad nota y que no se repita.

Buscando ayuda

Lo primero que quiero hacer es transmitiros mi agradecimiento por vuestra gran labor, gracias a vosotros podemos expresar nuestras dudas, peticiones y todo lo que necesitemos. Me llamo Alberto y tengo 17 años, estoy preparando un proyecto con un amigo, llamado Juan Luis. Se trata de un juego tipo *Golden Axe*, ambientado en la Edad Media, necesitamos ayuda en lo que podáis. Necesitamos tanto grafistas 2d como programadores, si os interesa poneros en contacto conmigo en la siguiente direccion: albertoblanc@yahoo.es. No nos importa si tengas experiencia en Div, lo importante son las ganas que le echemos. Dicho esto sólo queda despedirme y daros de nuevo las gracias por brindarme esta oportunidad. Hasta pronto.

Sonido 3D

Uno de los ganadores de nuestro número pasado nos ha hecho llegar este mensaje:

He recibido un mail de un Divero que al parecer ha creado un juego en el cual hay una función llamada Dolby, que hacía un efecto estéreo o algo parecido. Me dice que mi idea del sonido 3D es demasiado parecida a la suya.

Si habéis mirado todos los archivos del CD, veréis que en uno hay un mensaje que dice algo así como que pido perdón por si alguien ha hecho una cosa parecida, por que como no tenia ningún medio por el cual enterarme, no sabía si eso ya estaba inventado.

Aun así, también me gustaría que quedase claro que el sonido 3D fue una idea totalmente mía, sin ningún tipo de plagio de ninguna parte y que mi única intención era dar a conocer una funcion que no está incluida en DIV, y que creo que es interesante para todo creador de juegos 3D. Nada mas, perdón por las molestias y gracias por todo.

Error de DIV 2

En el número pasado en esta misma sección respondíamos a un lector que nos preguntaba sobre un error de DIV al que no sabíamos responder. Pues bien, poco después, él mismo nos contó que ya había encontrado la solución que os reproducimos ya que consideramos de interés para todos:

El problema sólo existe cuando se le da a una variable el valor "con" dentro del código del programa. Hay 2 formas al menos de resolverlo:

1º. Si la variable es tipo STRING le das el valor "co" y luego le añades una "n" o le pones "conw" y borras un carácter.

2º. La otra opción es cargarlo de un archivo externo con las funciones fopen() y fread(), teniendo en cuenta que se dan las órdenes escribiendo. Por eso necesitaba la palabra "con", porque es una preposición muy útil :•).

De todas formas el DIV2 tiene más "errorcillos". Está el del sonido... que se oye MUUY bajo... y los módulos de música se oyen MUUUY altos. Además está la función roll_palette() que en DIV1 funciona pero no en DIV2. El creador de DIV nos dijo que eso se solucionaba haciendo un fade_off antes() de roll_palette() y luego un fade_on(). Claro que hay que hacerlo muy rápido para que no se note... y es mejor usar fade (100,100,99,64) para que el cambio sea lo más rápido posible. Pero no son los únicos. Hay un problema al usar unload_song() ya que no descarga entre 32 bytes y 1 kbyte de memoria del módulo. Además las funciones de red no he conseguido que me funcionen nunca ¡eh, esto me interesa, ¿sabes de alguien que sí le hayan funcionado?! *José Luis Cruz*

Anuncio

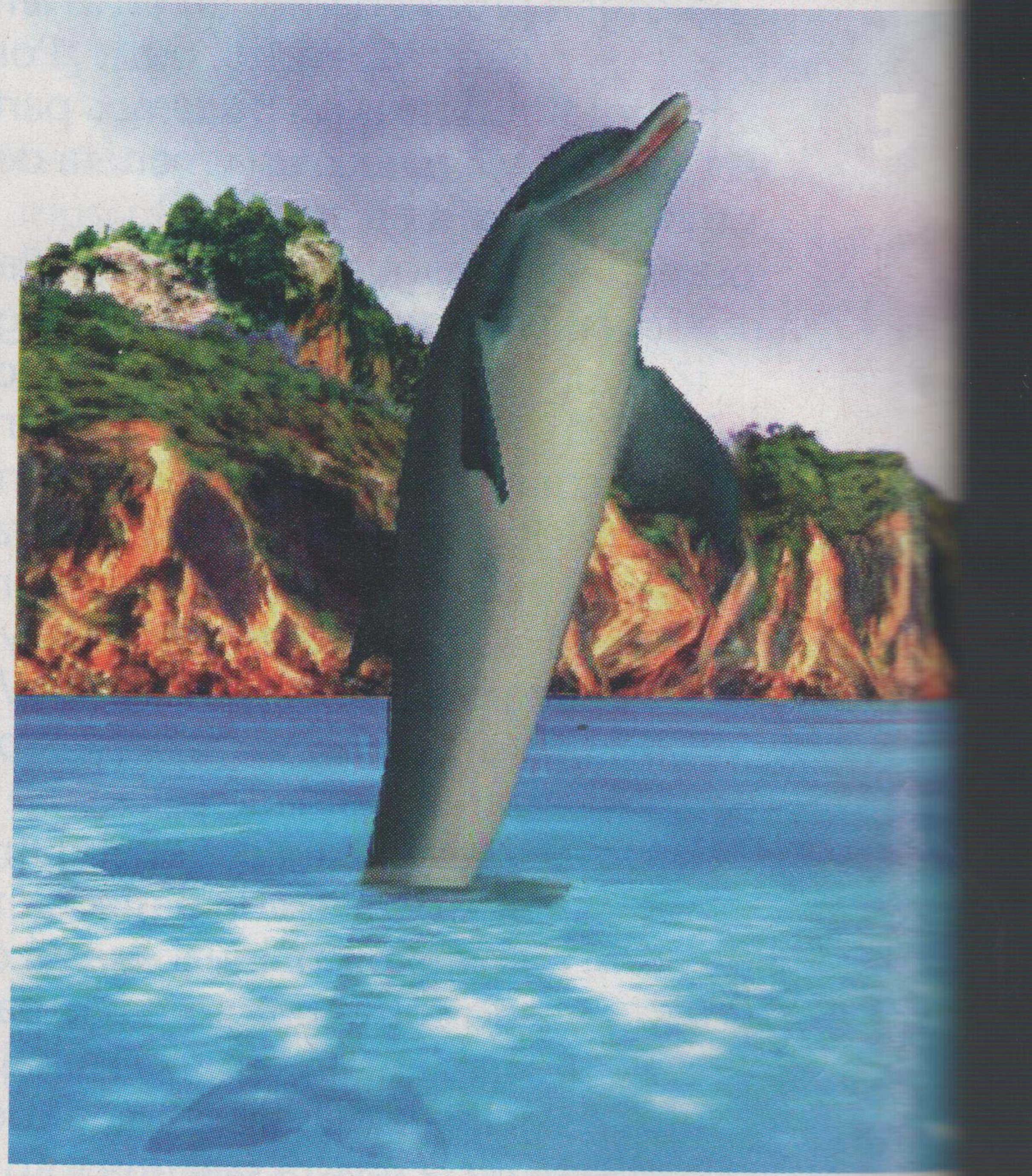
Busco Programadores Div. Soy de Santiago de Compostela; osea que prefiero que tú también lo seas. Yo soy músico, programador y grafista. Ocasionalmente soy escritor y artista, y miembro de la corte superior de los "idiotas ilustrados". Propongo un tipo de juego alejado del argumento estúpido –salva el mundo–, y la sugestividad por otros medios más psicológicos que el típico charco de sangre.

José A. Cerqueiro Otero. R/ Valiña nº 9 Grixoa, 15898, Santiago de Compostela, La coruña.
budabyte@de.com

Un poco de todo

Saludos, estimados amigos de Divmania, porque supongo, que nos podemos considerar todos nosotros como amigos, ¿no? Bueno, a lo que iba. En primer lugar, daros la enhorabuena por la publicación en vuestro número 6 de esos cincuenta juegos de los lectores, algo, a mi parecer, muy necesario; ya que el publicar tan solo los tres ganadores era insuficiente, o al menos a mi me lo parecía. También quiero felicitar a todos aquellos que se han ocupado de realizar los diversos cursos. *Pelegrina.*

Gracias por tus felicitaciones. Respecto a tus críticas y sugerencias, decirte que tomamos nota. Sólo una cosa, como ya hemos dicho en otra ocasión Divmanía no tiene relación directa con Hammer Technologies con lo que tus preguntas sobre el futuro de Div no podemos responderlas y respecto a las posibilidades de mejora, recordarte que ésta es una revista pequeña y, creenos, hacemos lo que podemos.



Esperamos vuestros mensajes

Para cualquier sugerencia, duda o aclaración podéis mandar vuestras cartas o correos electrónicos a la siguiente dirección:
Divmanía
C/ Alfonso Gómez 42. Nave 1-1-2
Madrid 28037. España.
Tfno: 91. 304. 06. 22
Fax: 91. 304. 17. 97
E-mail: *divmania@prensatecnica.com*

PROGRAMAR JUEGOS ES COSA DE NIÑOS SI TE SUSCRIBES a Div Manía



Si deseas estar en la vanguardia del mundo de la informática, suscribirse a DIV MANÍA es un primer paso acertado porque...

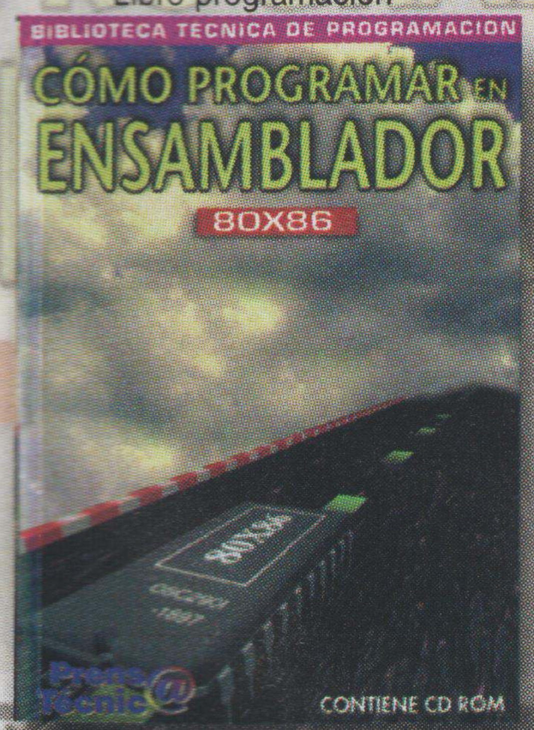
- *Es la única revista escrita por y para los programadores de videojuegos. Nuestra redacción está compuesta por veteranos desarrolladores, expertos del entorno DIV, grafistas y muchos otros profesionales del software de entretenimiento que dan lo mejor de sí a los lectores.*
- *Te ofrece lo último en el delicado campo de la programación de videojuegos, con los títulos que se encuentran en proceso y los productos recién salidos del horno o de los PCs.*
- *Para seguir avanzando hay que saber echar la vista atrás y a la vez no olvidarse del futuro, y **Div Manía** empieza un nuevo camino.*
- *Nuestra revista presenta un look muy cercano a sus lectores, salido de las inquietudes de todos vosotros.*
- *Nunca nadie te ha ofrecido tanto por tan poco; nunca has tenido tan cerca la oportunidad de estar al día de lo último en programación por el mínimo esfuerzo de acercarte al quiosco o enviar nuestro cupón y recibir la revista en casa puntualmente cada dos meses.*
- *En el interior del CD-Rom encontrarás los elementos con los que todos los programadores sueñan.*
- *Somos como tú y conocemos, más o menos, qué se esconde dentro de tu cabeza. Y si no lo conocemos aún, lo aprenderemos gracias a ti.*
- *Ofrecemos las más diversas sorpresas, las más interesantes ofertas, para que no te olvides de que la programación siempre está viva.*

Además, el suscriptor tiene derecho a la siguiente oferta:

- **Con un año de suscripción (seis números) regalamos un producto a elegir entre:**
  "Programación Gráfica para PC"
  "Cómo programar en Ensamblador"
- **Con dos años de suscripción (doce números) regalamos DIV 2**



PROGRAMACIÓN GRÁFICA PARA PC
Libro programación



CÓMO PROGRAMAR EN ENSAMBLADOR
Libro programación



DIV II
Creación de juegos

Solicite su ejemplar enviando este cupón por correo, por fax: 91 304.17.97 o llamando al teléfono 91 304.06.22 de 9:00 a 19:00 h.

CUPÓN DE SUSCRIPCIÓN ANUAL A DIV MANÍA

Deseo suscribirme a la revista **DIV MANÍA** acogiéndome a la siguiente modalidad:

☐ Suscripción: 1 año (6 números) por sólo 5.970 ptas. ☐ Correo certificado 1 año: 1.500 ptas. adicionales

☐ Suscripción: 2 año (12 números) por sólo 11.940 ptas. ☐ Correo certificado 2 años: 3.000 ptas. adicionales

Desde el número ☐

Además recibiré gratis:

☐ Por 1 año de suscripción: **uno** de los siguientes productos:
☐ Por 2 años de suscripción: **DIV II**

☐ Programación Gráfica para PC ☐ Cómo programar en Ensamblador

Nombre y apellidos ....
Domicilio ....
Población ....
C.P. .... Provincia ....
Telf. .... Profesión ....
DNI/NIF: ....

FORMA DE PAGO:

☐ Con cargo a mi tarjeta VISA nº________________ más gastos de envío
Fecha de caducidad de la tarjeta ....
Nombre del titular, si es distinto ....
☐ Domiciliación bancaria, más gastos de envío
Población ....
Ruego a Vd. que se sirva cargar en mi:

☐ cuenta corriente
☐ libreta de ahorro número

ENTIDAD	OFICINA	DC	Nº CUENTA

el recibo que será presentado por PRENSA TÉCNICA S.L. como pago de mi suscripción a la revista **DIV MANÍA** más gastos de envío.

FIRMA:

☐ Contrarreembolso del importe más gastos de envío.
☐ Cheque a nombre de PRENSA TÉCNICA, que adjunto más gastos de envío.
☐ Giro Postal (adjunto fotocopia del resguardo) más gastos de envío.

Prens@ Técnic@
de libros y publicaciones

C/ Alfonso Gómez, nº 42 Nave 1-1-2. 28037 Madrid. Tfno: 91 304 06 22. Fax: 91 304 17 97
e-mail: maspc@prensatecnica.com http://www.prensatecnica.com

Contenido CD-Rom



En este número nueve el contenido del CD-Rom que acompaña a la revista seguro que os será de utilidad para vuestros trabajos y proyectos. Aparte de los juegos ganadores y los complementos de los artículos, como es usual, tenemos unas cuantas demos y programas que seguro que usaréis.

JUEGOS GANADORES

Este mes se han presentado algunos menos juegos a concurso, después del verano seguro que nos llegan más. De todas formas, entre los juegos que nos han llegado a la redacción había tres que eran de obligado premio aunque eso no signifique que otros no lo merecieran. Aquí tienes los títulos que se han alzado con los laureles de la victoria, disfrútalos.

Ciclis

Un simulador de ciclismo ¿original, no? Desde luego a nosotros nos lo ha parecido y mucho. Pero es que además derrocha sentido del humor (y algo de mala leche, todo hay que decirlo) y, por tanto resulta muy divertido.

El secreto de la Isla Tortuga

Un misterioso secreto que tú solo debes desvelar valiéndote de tu inteligencia. Unos novatos en el tema se lanzaron a crear una aventura gráfica y ¡mirad el resultado!

Yamis Arena

El primer juego Div que funciona en red. Una auténtica exclusiva si no fuera porque conseguir que funcione en red no es nada sencillo.

DEMOS Y PROGRAMAS

Dreamweaver 3

Demo de este programa profesional que sirve para crear y mantener páginas web. Su completo menú de opciones y su fácil uso te permitirán diseñar la página donde meter toda la información sobre tus juegos o temas que más te interesen. Puedes hacer páginas completas usando HTML dinámico, meter imágenes, etc., sin escribir una sola línea de código o programando las páginas como tú quieras.

Fireworks 3

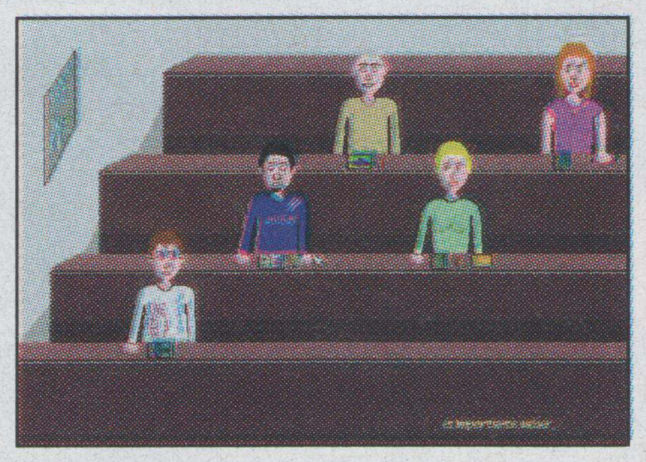
Diseña el aspecto gráfico y las imágenes que quieras meter en tu web con la demo de la última versión de este programa. Podrás encontrar múltiples herramientas de edición y construcción de todo tipo de imágenes con completos menús de ayuda que te hagan más fácil la construcción de una página web con un acabado impecable.

Dark Basic

Esta herramienta es un programa que hace más fácil usar BASIC. Dark Basic no requiere ningún conocimiento de programación, sólo hace falta imaginación para crear el software que tengas en la cabeza, desde simples editores de textos a juegos a pantalla completa y en tres dimensiones añadiendo sonidos, música, animaciones, textos y gráficos





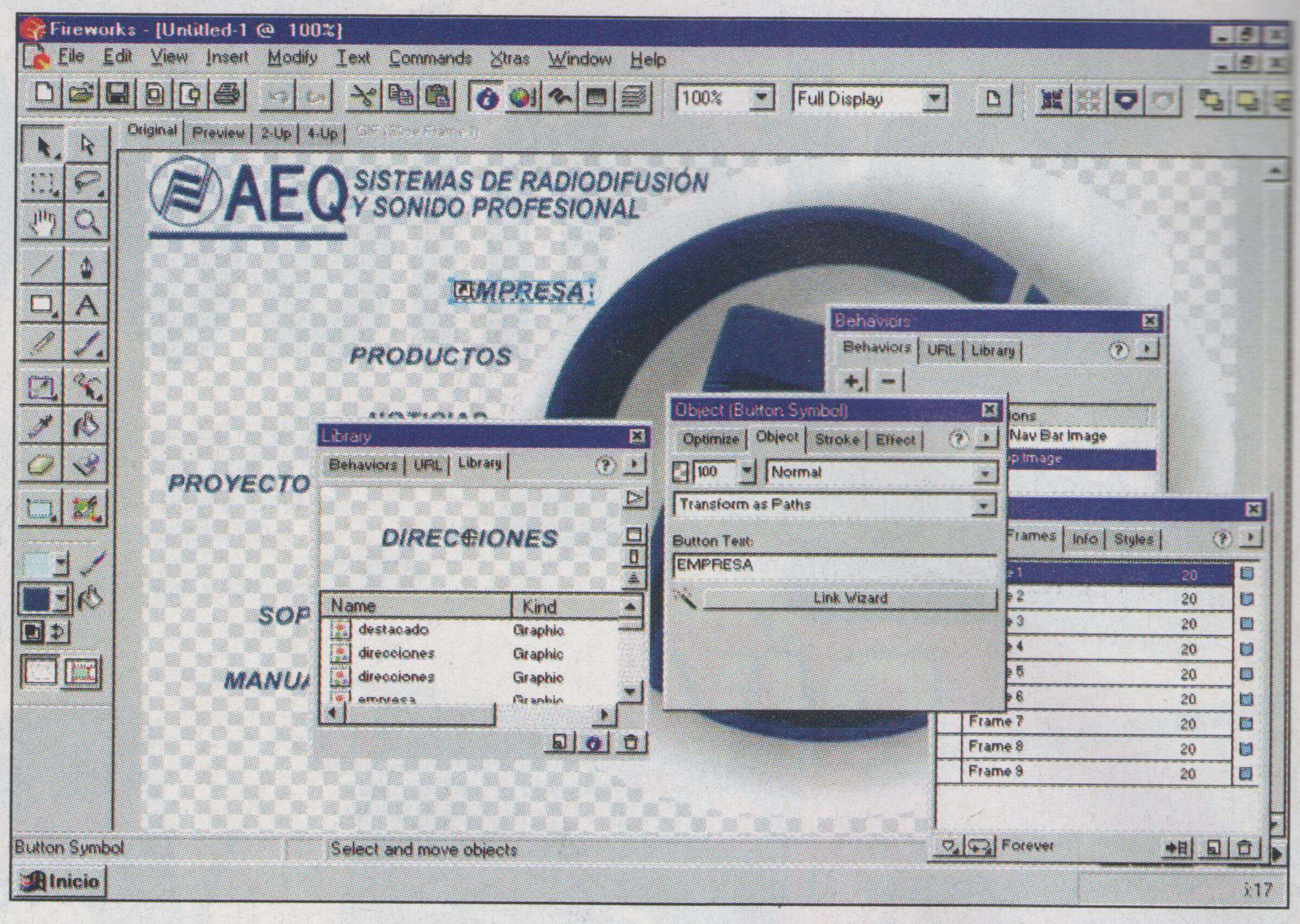




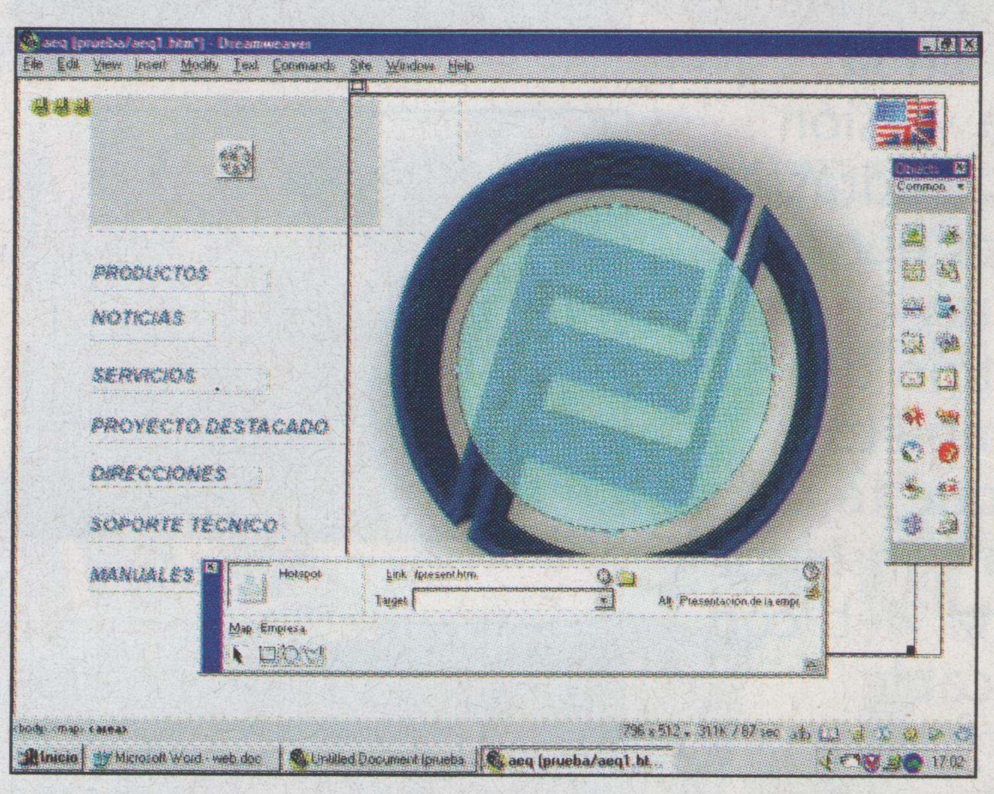


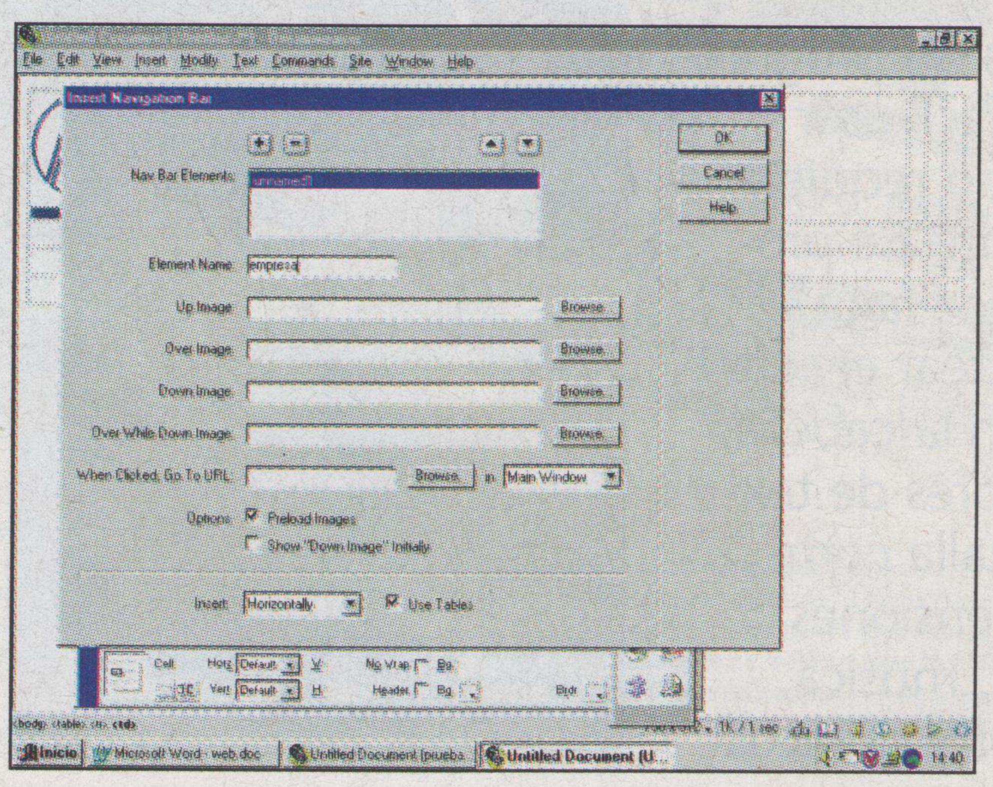






de manera sencilla. Este programa y los recursos gratuitos que vienen incluidos con él se ofrecen en exclusiva para los lectores de DIVmanía y no puede redistribuirse con fines comerciales.





M.U.G.E.N.

Con este programa podrás practicar fácilmente lo explicado en el artículo que aparece en la revista y diseñar los luchadores que quieras manejar en este beat´em up multijugador. También puedes crear los movimientos de combate que quieras que hagan y su animación respectiva. Para aficionados al noble deporte de la lucha cuerpo a cuerpo.

PROGRAMAS DE REGALO

CrannFénix

CrannFénix es un entorno de desarrollo visual basado en *Fénix*. Permite la creacción y el mantenimiento de proyectos en *DIV*, haciendo uso del compilador *Fénix* y de todas sus mejoras sobre el lenguaje original. Aporta un entorno unificado donde gestionar todos los objetos que componen un proyecto, desde los programas encadenados a los FPG´S,

mapas, vídeo, etc., todo bajo entorno Windows.

Editor de textos

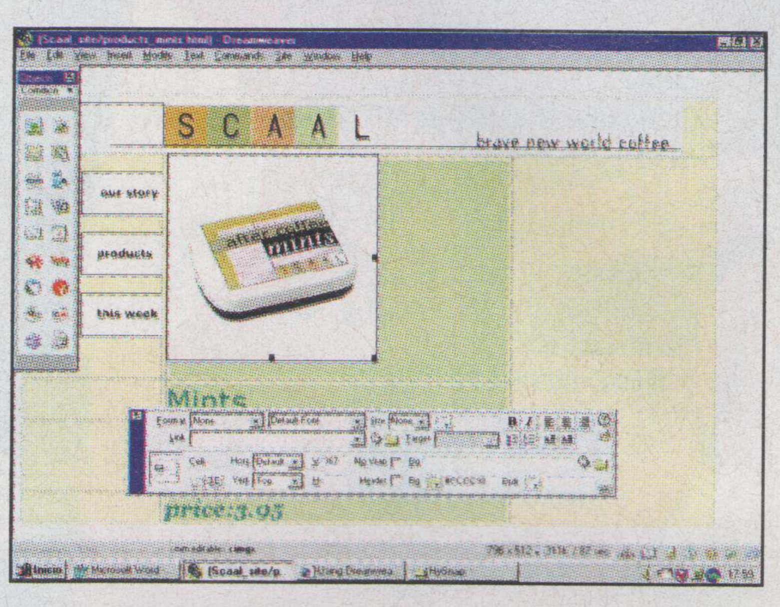
Un práctico editor de textos que nos ha ofrecido su autor para compartirlo con todos los lectores de DIVmanía.

Shareware

Este mes os ofrecemos *Palette Works* que es un programa que seguro que agradecerán los que se dedican a hacer juegos con DIV. Además podréis encontrar una completa colección del share más actual.

LIBRERÍAS

Aquí encontraréis unas cuantas librerías para que las uséis en vuestros juegos. Entre ellas, gráficos web que seguro sabéis aprovechar convenientemente en la realización de viestros nuevos proyectos y lo juegas con DIV.

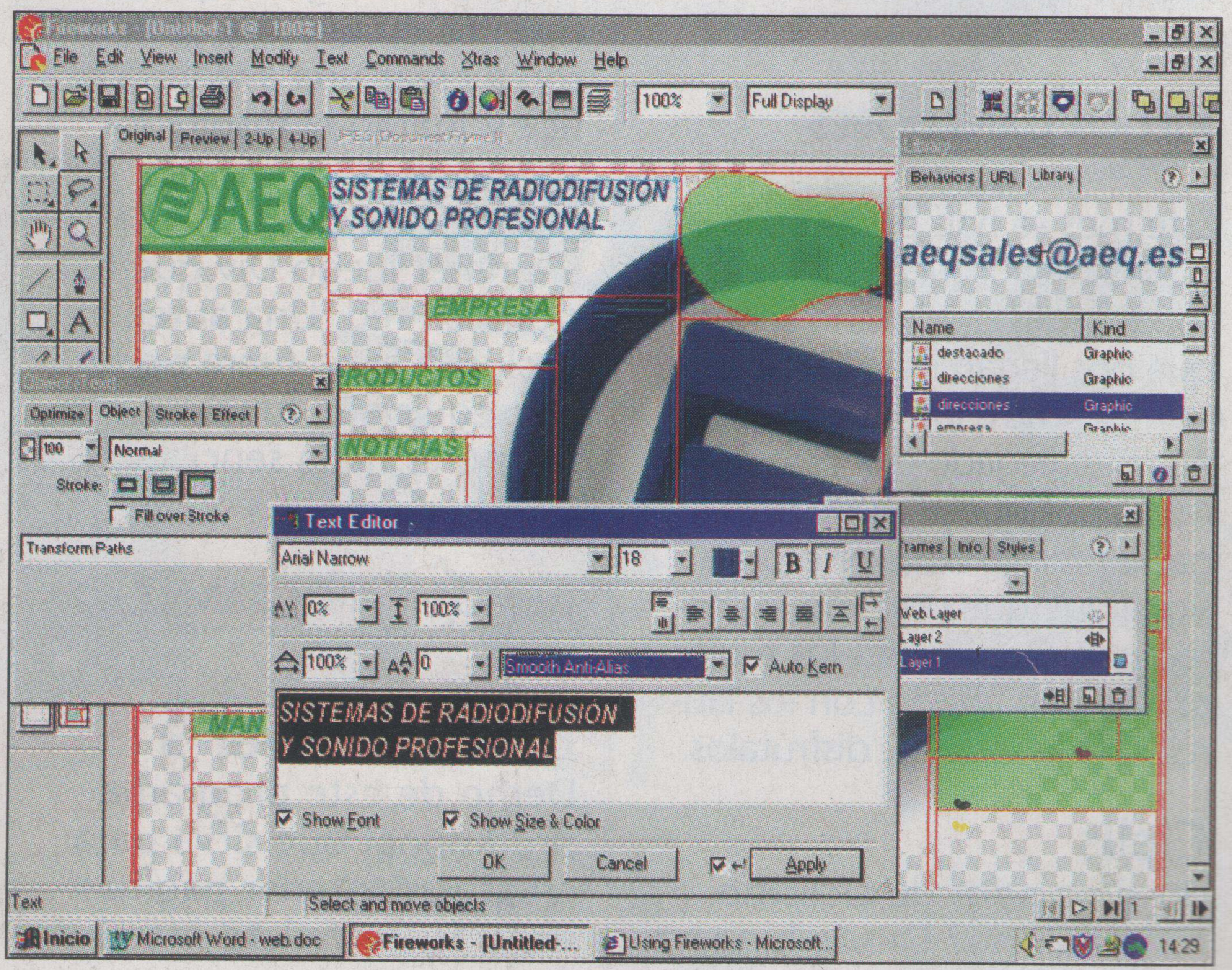


REVISTAS ELECTRÓNICAS

El número cuatro de la revista de Internet dedicada a DIV, Divnet, para todos aquellos que no puedan acceder a Internet.

CURSOS DE JUEGOS

Todos los complementos que necesitas para seguir todos las lecciones de nuestros cursos. Casi todas las secciones llevan códigos que por su extensión no es posible meter en la revista. En el Cd-Rom los encontrarás enteros y podrás seguir las explicaciones comodamente.





Space
Bunnies
La nueva heroína de los juegos 3D
MUST DIE
Sólo
18,00 €
PC CD rom
2.995
Ptas
De venta en quioscos, grandes superficies y tiendas especializadas
Manual en castellano
PC CD rom
COMPATIBLE
WINDOWS 98
ripcord
GAMES
Digital Dreams
MULTIMEDIA
Para más información, llamar al teléfono +34 91 304 06 22
Digital Dreams Multimedia
C/ Alfonso Gómez, 42, nave 1-1-2
28037 Madrid (España)
Fax: +34 91 304 17 97
www.spacebunnies.com

CONTENIDO DEL CD ROM



HERRAMIENTAS PARA TUS JUEGOS

DEMOS

Dreamweaver 3
Demo de este conocido programa de creación de páginas Web.

Fireworks 3
Diseña el aspecto gráfico y las imágenes que quieras meter en tu web con la demo de la última versión de este programa.

Dark Basic
Esta herramienta hace más fácil usar BASIC. Dark Basic no requiere ningún conocimiento de programación, sólo hace falta imaginación para crear el software que tengas en la cabeza.

M.U.G.E.N.
Un práctico programa para los amantes de los juegos de lucha.

JUEGOS GANADORES

Este mes son Ciclis, El secreto de la Isla Tortuga y Yamis Arena. Muchas felicidades a todos.

CRANN FÉNIX

CrannFénix es un entorno de desarrollo visual basado en *Fénix*.

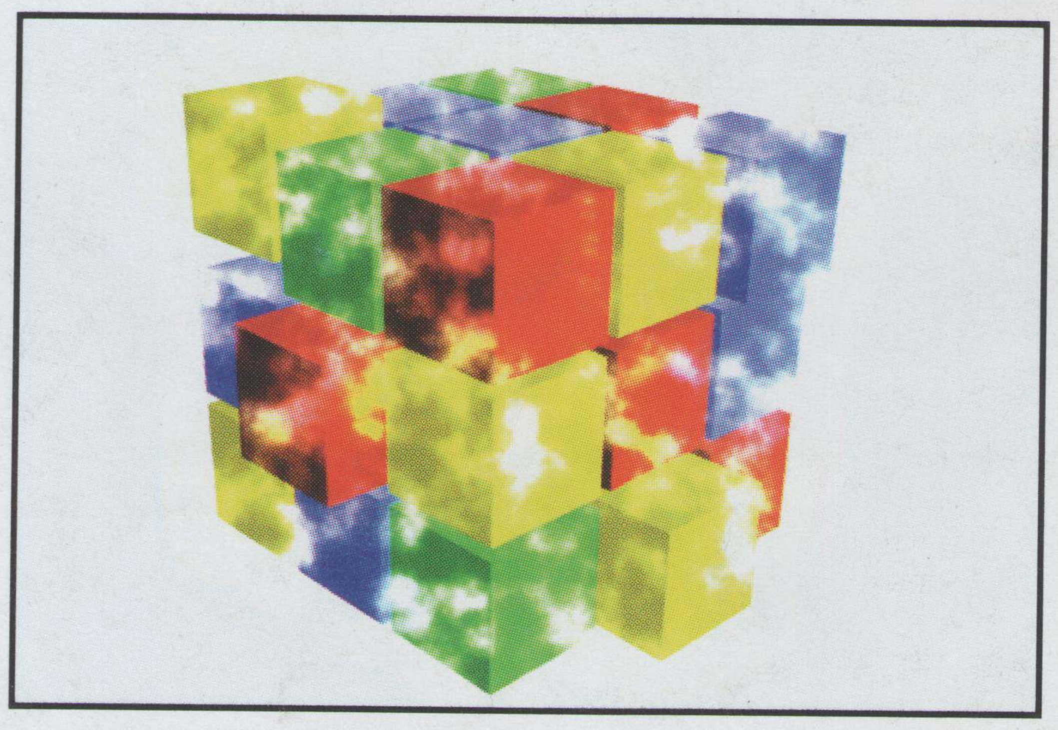
SHAREWARE

Este mes os ofrecemos Palette Works , un programa que vosotros nos pedistéis, además de una completa colección del share más actual.

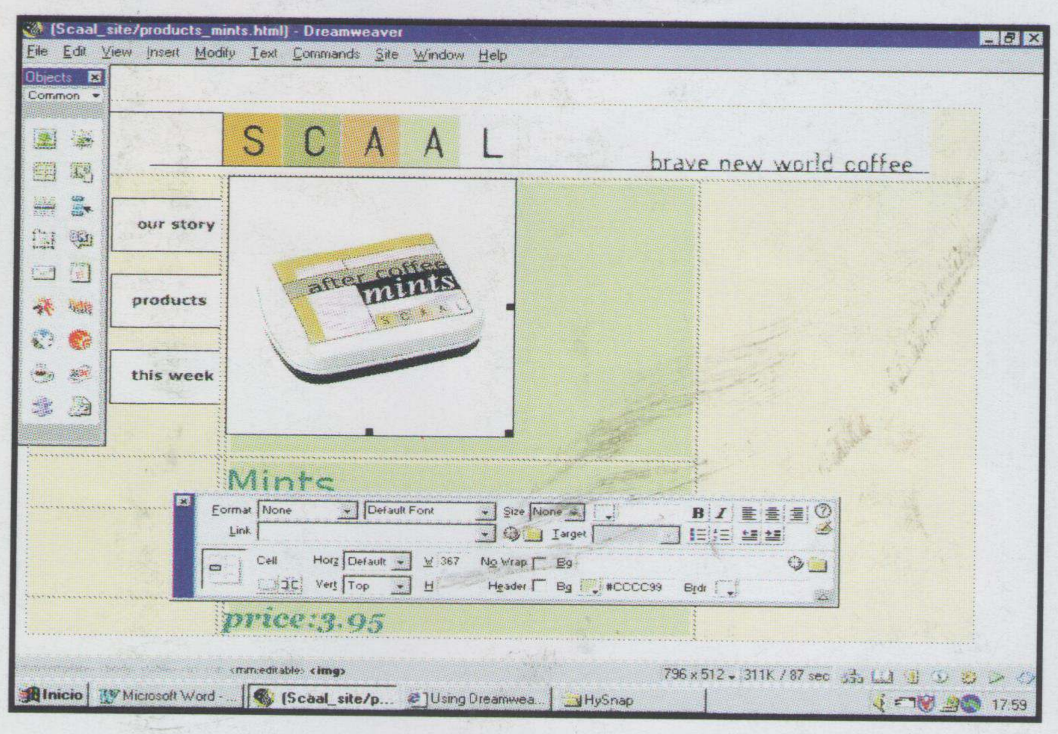
Revistas electrónicas

El número cuatro de la revista de Internet dedicada a DIV, Divnet, para todos aquellos que no puedan acceder a Internet.

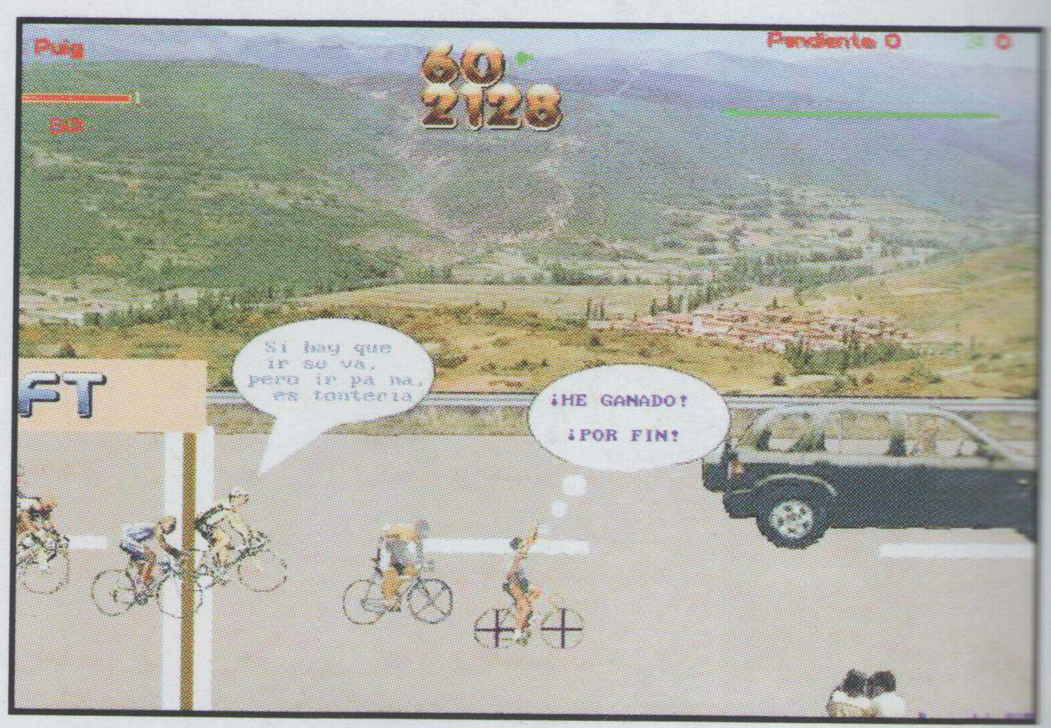
REPORTAJES. Todo sobre Stratos, una asociación de desarrolladores que a buen seguro conoceréis.



PROGRAMAS. Herramientas para el diseño de páginas Web.

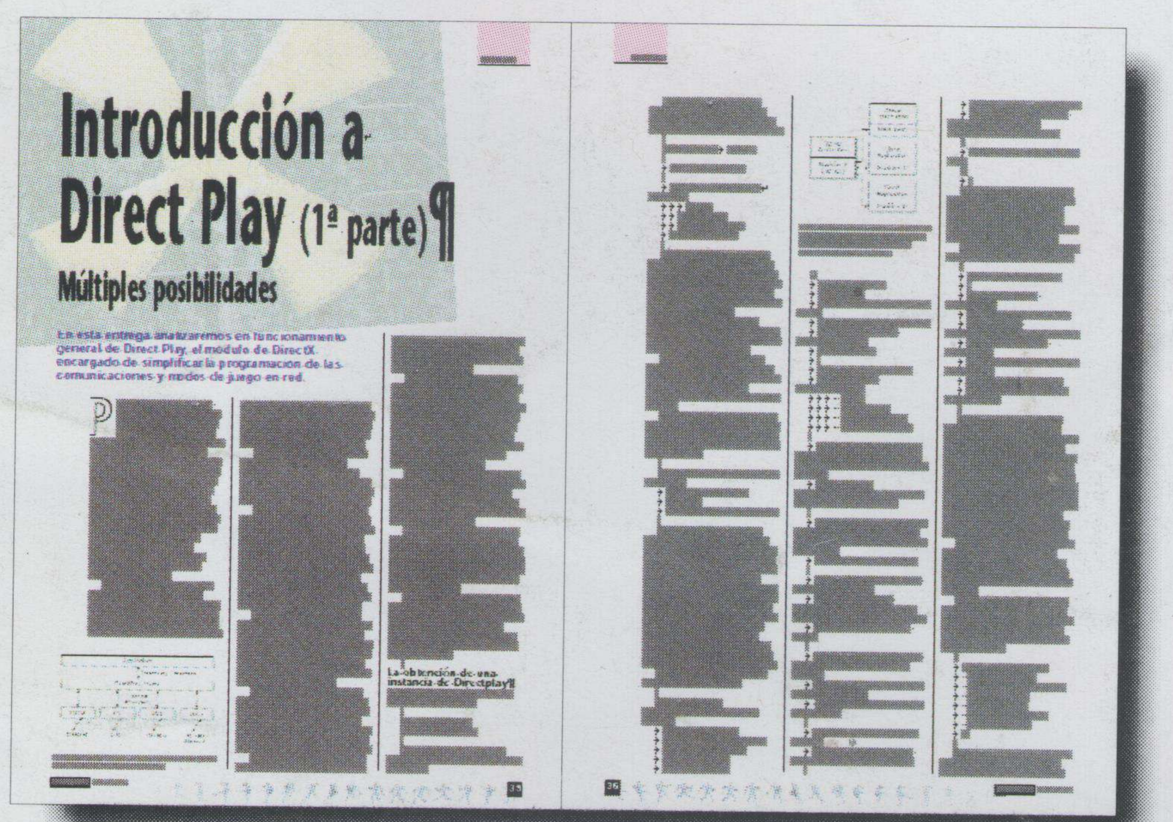


CONCURSO DE JUEGOS. Los afortunados juegos que se han alzado con la victoria.



DIV manía CON EL MEJOR CONTENIDO


Stratos
Un proyecto con futuro


Introducción a Direct Play (1ª parte)
Múltiples posibilidades


M.U.G.E.N.
O cómo hacer tu propio juego de lucha

ACTUAL — EXHAUSTIVO — DIDÁCTICO

Y MUCHO MÁS...